

CRAM



LINUX ESSENTIALS

(010-160)

*A Time Compressed Resource
to Passing the LPI® Linux Essentials Exam on Your
First Attempt*

JASON DION

Copyright © 2020
Dion Training Solutions, LLC
www.DionTraining.com

All rights reserved. Except as permitted under United States Copyright Act of 1976, this publication, or any part thereof, may not be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, storage in an information retrieval system, or otherwise, without express written permission of Dion Training Solutions.

Manuscript edited by Mr. John Himes

ISBN: 9798666362525

DISCLAIMER

While Dion Training Solutions, LLC takes care to ensure the accuracy and quality of these materials, we cannot guarantee their accuracy, and all materials are provided without any warranty whatsoever, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose. The name used in any data files provided with this course is that of a fictitious company and fictional employees. Any resemblance to current or future companies or employees is purely coincidental. If you believe we used your name or likeness accidentally, please notify us and we will change the name in the next revision of the manuscript. Dion Training Solutions is an independent provider of integrated training solutions for individuals, businesses, educational institutions, and government agencies. The use of screenshots, photographs of another entity's products, or another entity's product name or service in this book is for educational purposes only. No such use should be construed to imply sponsorship or endorsement of this book by nor any affiliation of such entity with Dion Training Solutions. This book may contain links to sites on the Internet that are owned and operated by third parties (the "External Sites"). Dion Training Solutions is not responsible for the availability of, or the content located on or through, any External Site. Please contact Dion Training Solutions if you have any concerns regarding such links or External Sites. Any screenshots used are for illustrative purposes are the intellectual property of the original software owner.

TRADEMARK NOTICES

LPI® is a registered trademark of the Linux Professional Institute in the United States and/or other countries. All other product and service names used may be common law or registered trademarks of their respective proprietors.

PIRACY NOTICES

This book conveys no rights in the software or other products about which it was written; all use or licensing of such software or other products is the responsibility of the user according to terms and conditions of the software owner. Do not make illegal copies of books or software. If you believe that this book, related materials, or any other Dion Training Solutions materials are being reproduced or transmitted without permission, please email us at piracy@diontraining.com.



To the best wife, friend, business partner, and supporter a husband could hope to have, and for her enduring patience with me as we continue our non-stop journey through life together.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

BONUS CONTENT

CHAPTER ONE

Introduction

CHAPTER TWO

Evolution of Linux

CHAPTER THREE

Open Source Software

CHAPTER FOUR

The Desktop Environment

CHAPTER FIVE

The Linux Shell

CHAPTER SIX

Linux File System

CHAPTER SEVEN

Searching and Extracting Data

CHAPTER EIGHT

Scripting Basics

CHAPTER NINE

Packages and Processes

CHAPTER TEN

Networking Basics

CHAPTER ELEVEN

User Accounts and Groups

CHAPTER TWELVE

Ownership and Permissions

CHAPTER THIRTEEN

Conclusion

CHAPTER FOURTEEN

Practice Exam

APPENDIX A

Answer Key to Practice Exam

LPI® LINUX ESSENTIALS

Exam Vouchers

ABOUT THE AUTHOR

ACKNOWLEDGEMENTS

This book is written for my community of students worldwide who have allowed me to continue to develop my video courses and books over the years. Your continued hard work throughout your careers continue to lead you upwards to positions of increased responsibility, and I am thankful to have been a small part of your success.

I truly hope that you all continue to love the Cram to Pass series and the method to my madness as you work to conquer the LPI Linux Essentials® certification exam. I wish you all the best as you continue to accelerate your careers to new heights.

BONUS CONTENT

This book comes with an accompanying video for specific topics. To view these videos, please scan the QR code in the title of that section to go deeper into these topics with some video demonstrations in our lab environment.

Please visit <https://www.diontraining.com/LEBook> to register your book and receive access to additional online practice exams.

CHAPTER ONE

Introduction



OBJECTIVES

- Understand how this book is designed to help you quickly pass your certification exam
- Understand how the exam is designed and how to take the certification exam
- Understand some tips and tricks for conquering the LPI Linux Essentials® certification exam

In this book, you will receive a crash course that will introduce you to everything you need to know to pass the LPI Linux Essentials® certification exam. This book covers just the essentials with no fluff, filler, or extra material, so you can learn the material quickly and conquer the certification exam with ease.

The LPI Linux Essentials® exam is the first certification exam in the Linux Professional Institute's certification path. This certification is designed to test your ability to use the basic console line editor and to demonstrate an understanding of processes, programs, and components of the Linux operating system.

This book assumes that you have no previous experience with the Linux operating system and will teach you exactly what you need to know to take and pass the Linux Essentials® certification

exam on your first attempt.

Now, this book will NOT teach you everything you need to know to be efficient or effective in utilizing Linux on a daily basis. After all, the LPI Linux Essentials® certification is designed to introduce you to the Linux operating system and its various concepts. If you want to become an expert in Linux, then this book and the LPI Linux Essentials® certification are a great start but remember that they are both written at the introductory level. Once you finish them, I recommend continuing upward to the LPIC-1 (System Administration) certification and its associated video courses and textbooks to further enhance your skills. This text singularly focuses on getting you to pass your certification exam, not to make you an expert in the Linux operating system, its system administration, or Linux engineering.

Due to the design of this text, we will move at a very quick pace through the material. If you read this entire book and take the practice exam at the end of the text (scoring at least an 85% or higher), you should be ready to take and pass the LPI Linux Essentials® certification exam on your first attempt!

Exam Basics

The Linux Essentials® certification exam is an entry-level certification for Information Technology personnel interested in understanding the Linux operating system, its usage, and its operation. This introductory certification covers a general overview of the Linux operating system: how to install it, how to configure it, and how to operate programs within it, both in the graphical user interface (GUI) and the command line interface (CLI).

The target audience for the LPI Linux Essentials® certification is for those who need:

- An understanding of the Linux and open source industry
- An understanding of the major components of Linux work
- A basic knowledge of the command line
- A basic understanding of networking and security in Linux

The certification exam consists of 40 multiple-choice questions

which must be completed within 60 minutes. A minimum score of 500 points on a scale of 200-800 points is required to pass the certification exam, equating to a score of about 70% or higher. The exam is a closed book exam, with no notes or study materials allowed during your examination.

The current cost of the exam at the time of publication of this book is \$120 USD, and one can take the exam at any PearsonVue testing center worldwide or online with the OnVue service.

Dion Training Solutions (www.diontraining.com) is an LPI Approved Training Partner for the Linux Essentials® certification exam. To purchase your exam voucher at a special discounted price available to our readers, please visit diontraining.com/vouchers.

Exam Tips and Tricks

Before we dig into the contents of the LPI Linux Essentials® certification exam, it is important for you to understand some exam tips and tricks. This will help you to grasp exactly how to study for the exam as you read through the rest of this book, and it will help you to focus your efforts to get the most out of this material.

The most important thing to remember when taking the LPI Linux Essentials® certification exam is that there are no trick questions on test day. Every question is precisely worded to match the material you have studied. You should read each question multiple times to ensure that you understand exactly what it's asking and that you are answering the question being asked. Anytime you see the words ALWAYS or NEVER in an answer, think twice about selecting it. As in most things in life, rarely is there a case where something ALWAYS or NEVER applies to a given situation!

As you read the questions and answers, be on the lookout for distractions or red herrings. Generally, there is at least one of these listed in the possible answer choices to try and distract you from the correct answer.

If you see part of a question with bold, italics, or all uppercase letters, you should pay close attention to those words because the test writers decided that those keywords are important to selecting the correct answer.

It is important to remember what concepts in Linux and the open source environment were covered in your studies, since you will see questions on them again come test day. You shouldn't see

things on the exam that were not covered by the textbook or video course you used to study for the exam. We will cover all these concepts throughout this book, as these are the building blocks of Linux Essentials® curriculum and the certification exam. Also, if a question asks about a particular command, then make sure that you don't select an answer that contains a concept or process. Always be careful to answer the question being asked.

Also remember that you must answer the questions based on your Linux knowledge and studies from this textbook, not your personal workplace experience. Your workplace may implement Linux and its concepts differently in their situation and use case. When in doubt, you should always select the book answer when answering a question on the certification exam.

On exam day, you should also select the BEST answer. Each question may have several right answers, but one is always the *most* correct answer. When in doubt, choose the answer that is correct in the *greatest* number of situations!

On test day, you don't have to memorize the terms from this course on LPI Linux Essentials® word for word, but you must be able to recognize them from the choices given. During certification exams, you will choose your answer from a multiple-choice style question instead of a fill-in-the-blank or essay question. This is an essential difference in certification testing and the tests you may have taken in high school or college. In the certification world, you just need to be able to recognize, not regurgitate, the information.

As you study this book, keep these objectives in mind:

- Know terms associated with Linux and open source concepts
- Differentiate between a command and its arguments
- Be able to utilize the command line interface to perform various functions

CHAPTER TWO

Evolution of Linux



OBJECTIVES

- Understand the open source philosophy
- Describe various Linux distributions and their lifecycles
- Understand Linux's use in embedded systems
- Understand basic hardware requirements for Linux
- Describe differences between Linux and other operating systems

With the emergence of the internet and new technologies, Linux has become so common that it is a household name. Linux is a family of open source, Unix-like operating systems, typically packaged into a distribution. It is the operating system of choice for many educational and commercial institutions due its robustness and extremely low cost of acquisition. Linux distributions include the Linux kernel and supporting system software and libraries to provide features that users interact with.

You may have heard of popular Linux distributions such as Ubuntu, Debian, and Fedora, but there are hundreds, if not thousands, currently in active development. Linux's popularity is mainly because it is open source software, which allows anyone to freely download, modify, and even redistribute it. The open source origin of Linux is one of the main reasons for its widespread

adoption and success.

Most Linux distributions include a windowing system and a desktop environment that provides the user with a graphical user interface (GUI), much like Microsoft Windows. We will cover all the major features and functions of this wonderful operating system throughout this book.

There are many Linux distributions that are solely for servers. These distributions often omit the graphical desktop altogether. This means that it is important for you to also become familiar with the Command Line Interface (CLI) or terminal. This command shell requires proper syntax to utilize, though, and is considered harder to use than the GUI alone.

For this reason, it is important for you to learn both the graphical interface and the command line interface to become proficient in Linux. If you want to become a successful end user or Linux system administrator, it is worth your time to learn to use both environments adequately.

But why should you want to learn how to use Linux anyway? Well, if you take a quick look at the internet, you will find that Linux is everywhere. In fact, over 96% of all web servers operate on servers running one of the numerous Linux distributions. Linux is considered the go-to operating system in many industries due to its advantages, like scalability and open source, over other operating systems.

Open Source Philosophy

You may have encountered the term open source while browsing the internet or participating in some online forums. Linux is heavily influence by the open source philosophy. This mindset has had an extreme impact on development of the Linux operating system and its numerous distributions.

Open source refers to computer software or programs in which the source code is readily available for public use or modification from its original design. It encourages a collaborative effort among programmers and general users in that program or software's community to improve on its design and purpose.

What are some examples of open source software? Well, if you just look at a simple web server, you will find a lot of open source software. For example, if you are running an Apache web server, then you are running open-source software. How about WordPress?

Well, WordPress runs over 30% of the internet's websites these days, and guess what? It is also open source.

Since WordPress is open source, this means that you can go to WordPress.org, download the entire code base for the WordPress content management system, and you can modify, edit, or change it as much you like. This is the great thing about open source software. You have full access to all the code that runs these programs, and you can modify it to meet your exact needs.

The software or program's original designers then release the source code under the terms of a software license. The software license will dictate what users can modify in the original package distribution and how they can redistribute the modifications (called versions or forks) to the community and general public.

There are many different types of software licenses used in the open-source community. These include the GNU General Public License, the Apache License, the MIT License, and even the Unlicense. These licenses represent the entire spectrum of open source licenses, from highly protective to unconditional. Depending on your specific needs for your open-source project, you can choose which of these open-source licenses work best for your use.

Linux Distributions

We already discussed that Linux is packaged into several distributions (distros for short). Each distro contains a Linux kernel, supporting software and libraries, as well as configuration files. By combining all these components, Linux becomes a complete operating system, just like Microsoft Windows or Macintosh OS X.

Linux distributions differ from one another depending on who the developers are and what they want their distribution to achieve. Almost like a brand-new company, each distribution has its own goals and mission. Based on this, some distributions are more popular than others and can server larger or smaller audiences.

At the core of any Linux distribution is a Linux kernel. A kernel is a low-level computer program functioning as the bridge between the user and the computer's resources. Some of its functions include memory management and managing input and output devices. Essentially, the Linux kernel is the core computer program that has complete control over everything in the system. As the Linux kernel is constantly evolving, two distributions are likely to use slightly different kernels, as the developers make small changes to fix bugs

or to add features and functions to their own distro's kernel.

In addition to the kernel, some distributions also come packaged with different software and tools such as the X Window System (called X11 or X), as well as utilities to manage disks that are critical to the system's normal functioning. The X Windows System is an example of software providing the basic framework for the graphical user interface (GUI) environment. This allows Linux users to draw and move windows on the monitor and to interact with a mouse and keyboard, similar to what you are used to when you use a Windows or Mac computer.

Since a Linux distribution is a complete operating system, additional software such as server daemons, networking programs, desktop environments, and productivity tools are also shipped as part of the complete system. This supplemental software helps to provide specific types of branding for some distributions. This is especially true when it comes to the desktop environment and the ability to easily manage the system. For example, a specific distribution may come with the commonly used productivity software like word processing, spreadsheet, or presentation programs already installed.

One unique feature of Linux is the way the system manages startup processes. Different distributions use different scripts and utilities to launch programs that link the computer to a network, present the login prompt, and other common functions. This gives each distribution a unique "personality," which one can configure according to the user's specific preferences.

Typically, Linux distributions are available for download from their developers' websites. You can download image files, which we then burn onto a CD or DVD, or, with the help of a simple program, we can simply use a USB flash drive as the installation media. Large companies and commercial users can also install the distribution directly onto a private virtual server or a commercial cloud services, such as those offered by Amazon Web Services (AWS), Google Cloud, or Microsoft Azure.

Distribution Lifecycle

Linux developers manage their distribution's lifecycle through release schedules. Release schedules specify when new versions of a distribution are released to the public. Generally, only the final release version is recommended for use since it is the most stable.

However, developers can also publish versions that are recommended only for testing and debugging, called pre-release versions. These versions are categorized as either alpha or beta releases. An alpha release contains the newest features, but they also can contain the most bugs. So, you should never use an alpha release of a product in a production environment since it can fail easily and often. A beta release tends to be more stable, but remember it still contains code that is being tested and therefore bugs can occur. Again, in a production environment, it is better to wait until after beta testing is complete before installing a particular version of a distribution. When the release version is released by a developer, it is considered to be stable.

Most release schedules are publicly announced months or years in advance. However, security enhancements, debugging, and other factors may cause delays. This practice is generally acceptable, if the delay does not take too long. If there is an excessive delay, it can cause end users to question whether the developers abandoned the project.

Now, I know this sounds strange, but it is actually something that people do worry about. This is because developers abandon projects all the time. Many smaller distros are created by just one or two people, and, since they are usually put out for free, a developer may simply abandon the project if they don't have the time to work on it anymore. After all, most of the smaller distro developers do so as a hobby, for fun, or as a labor of love. If their life circumstances change, sometimes they stop developing their distro. For this reason, I'd try to stick with the major and well-known distros for any Linux systems you plan to run in a production environment.

When researching different distros, you will see that the developers often come up with catchy names for their different versions. This is helpful for referring to a specific version, especially compared to the usual number-point-number scheme. These catchy names also differentiate a major version change from a minor one.

For example, Ubuntu uses a two-word naming convention that uses an alliterative adjective-animal pair which starts with the first letter of the animal being described. The current version of Ubuntu as of this writing, is 19.04, also known as Disco Dingo. The previous version, 18.10, was called Cosmic Cuttlefish.

After the release of a certain version, the developer typically supports it for specific period. This time-period will vary based on

the distribution itself and can be anywhere from a few months to a few years. During this period, developers will regularly provide updates to improve performance and features, as well as patches to fix bugs or vulnerabilities in the system's security. You can continue to use that version beyond its end of life support date, but the developers will no longer supply updates and patches. This may be fine if you can patch the system on your own, but this will require you to compile it from the source code and search for a solution from that version's other users to continue to support it past its nominal end of life. Instead, I find it is a better idea to regularly update your system and upgrade it to the next version before reaching the end of life date.

Some distributions also offer different lengths of support, such as short-term support versions and long-term support (LTS) versions. We usually favor LTS versions over short-term support versions since they have more stability. By using a long-term support version, you can minimize disruptions and costs associated with upgrades if you are using them in a production environment. For example, Ubuntu provides up to 5 years of support to their LTS versions, but only provide 12 months of support for their non-LTS versions.

To upgrade a distribution, some require an administrator to download an image of the new version and install it as if it were a new operating system. However, this type of upgrading is time-consuming and requires a lot of user intervention. It is also prone to errors and data loss since it requires reformatting the system.

For an easier upgrade path, some distros have rolling release schedules, which means that upgrades occur in an ongoing manner. This eliminates the hassle of installing the new version from the beginning each time an upgrade is released. If you are coming to Linux from a Windows operating system environment, this is what you are already familiar with since Windows relies on this concept of rolling releases and updates to patch its security issues and add additional features over time.

Comparing Distributions

Scan QR code to watch a video for this topic



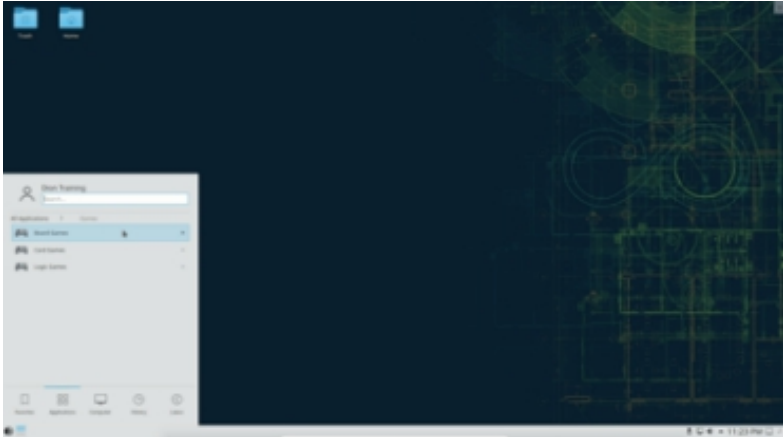
A few examples of different Linux distributions are Ubuntu, OpenSUSE, Debian, and Gentoo. While they are all Linux, they look and act differently. For example, Ubuntu comes with the Unity desktop by default. Unity is a well laid-out and graphically pleasing environment. Clicking on the dots at the bottom left corner of the screen brings up a selection of all the programs installed in Ubuntu. On the left portion of the screen, there is a column of icons, one of which is Files. Files brings up Ubuntu's graphical desktop file system, which is very similar to Windows Explorer. Pre-installed on Ubuntu is the web-browser Firefox, which allows the user to visit any website on the internet. Ubuntu also has an app store called Ubuntu Software, which allows a user to search for software such as audio and video, games, productivity and so much more. To install software, simply click on the icon of that software and follow the onscreen instructions.

OpenSUSE with the KDE desktop environment looks like the traditional Windows desktop, with the start menu on the bottom

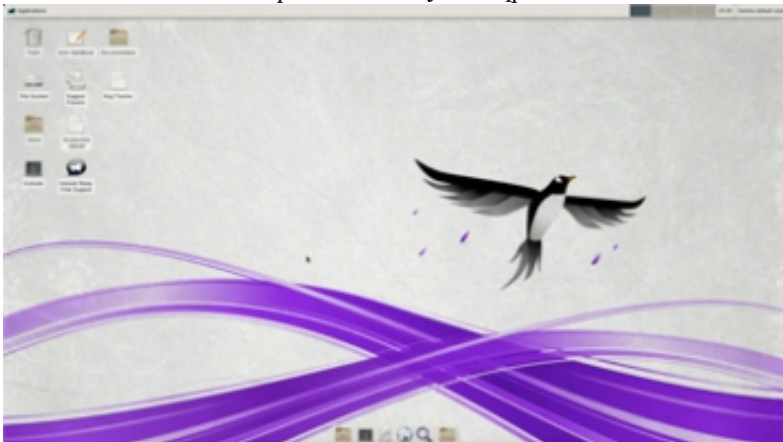
left corner of the screen. Clicking on that button shows items like the power button, your computer, and the applications. Just like in Windows, right-clicking on the desktop gives the option to change the background. OpenSUSE also comes with the LibreOffice office suite, which works just like Microsoft Office. LibreOffice allows us to create documents, spreadsheets or presentations.

The Gnome desktop running on Debian looks and behaves differently from Unity or KDE. Debian is known as a "flat" operating system because it is not graphically pleasing. To access the applications, the user can click on Activity at the upper left corner of the Gnome desktop to open a menu that shows some applications like Firefox, LibreOffice, or its own application installer. The point is that all of these are accessed differently from the two previous distributions.

Gentoo comes with XFCE. This desktop environment has a low system requirement and a low intensity desktop. It looks "flatter" than the other three desktops environments. Resembling a Macintosh system, it has an application launch bar along the bottom middle portion of the screen. Since Gentoo is a lower requirement distribution, it can run on older computers that had slower specs than the computers on today's market. In the XFCE desktop environment, there are several ways to access and run applications. Right-clicking on the desktop will show applications as the last option on that menu and expands to show the list of applications. The second way of launching applications on the desktop environment is by clicking the Applications button at the upper left corner of the screen. The magnifying lens located on the launcher at the bar at the bottom of the screen also allows us to search for the application we need to run.



OpenSUSE with KDE Desktop



Debian with GNOME Desktop

Comparing these four distributions just shows that using Linux is nothing to fear. This should hold true especially for users who may want to give it a try or are thinking of switching over from Windows or Mac because it operates similar to those two operating systems and, therefore, is something that they are already used to. Users will just have to decide which distribution and desktop environment suits their needs and preferences.

Embedded Systems

Up to this point, we have only spoken about Linux as a full-featured and complete operating system, but Linux is used for so much more than just computers. In fact, Linux is at the core of many of the devices that you use every day because it serves as the

core of many embedded systems.

An embedded system is a controller with a dedicated function within a larger mechanical or electrical system. It is embedded as part of a complete device, often including hardware and mechanical parts. As a matter of fact, embedded systems control many of the devices you use every single day, whether you realized it or not. These devices include, but are not limited to, industrial automation, navigation equipment, medical devices, and many others.

In recent years, due to its low cost and ease of customization, Linux has been shipped in many consumer devices such as wearable technology, home appliances, automobiles, and even thermostats. If you have a smart thermostat in your home, like a Nest® or an ecobee®, you already have Linux installed in your home!

Some embedded systems, like the Arduino and Raspberry Pi, are small credit card sized embedded systems that we can use to create other complete systems. These include things like robots and ultraportable computer systems. The Arduino and Raspberry Pi are available in the market as do-it-yourself or DIY kits. The Linux-based operating system embedded in these kits has a small footprint, requires little memory and processing power to run, and is highly customizable.

Most smartphones and tablets also run on a Linux kernel-based operating system known as Android. Google acquired Android in July 2005, extended it, and as a result Android is a highly competitive mobile operating system that now has a market share consisting of over 75% of all smartphone and tablet users worldwide. Android is highly scalable, user friendly, open source, and free for manufacturers to use within their devices.

Because of this, Android has even been ported for use in media players, televisions, and automobile entertainment systems, too. Android is highly customizable and is a truly complete operating system with Linux at its core. We can also add applications into the Android environment to further enhance features. There are applications for media content such as photos, videos, and social media, as well as productivity and gaming.

In fact, since many newer televisions have Android installed within them as an embedded system, these devices are now be complete multimedia centers, and we can use them for internet video streaming, browsing, and video games.

Hardware Requirements

Before you can install Linux, you need to know which hardware components you need to run and use it efficiently. These prerequisites are known as system requirements and are often a guide as opposed to an absolute rule.

The most common set of system requirements defined by any Linux distribution is the physical computer hardware. A hardware requirements list is often accompanied by a hardware compatibility list (HCL). An HCL lists tested, compatible, and sometimes incompatible hardware devices for a particular distribution. Most distros define two sets of system requirements, the minimum requirements and recommended requirements. These system requirements also tend to increase over time with a demand for higher processing power and resources in newer versions of each distribution.

Usually, developers will publish system requirements for a specific version of their distro on the download page of their website. These lists typically state required processor speed, memory size, available hard disk drive space, optical or removable media, display specifications, and other peripheral devices needed.

The minimum system requirements are the lowest possible hardware that your computer needs to boot Linux successfully and use it with basic functionality. As an example, Disco Dingo, the current version of Ubuntu as of this writing, requires a computer with at least a Dual Core processor running at 2GHz, 2GB of RAM, and 20GB of hard disk space. Most modern desktops and laptops meet these requirements without too much difficulty.

But minimum requirements can also vary greatly from one distribution to another. For example, consider the AntiX distribution. This distro only requires a computer with at least a Pentium III processor, 256Mb RAM, and 2.7GB of hard disk space. These requirements are extremely lightweight, considering that the Pentium III processor was first manufactured in 1999. It is pretty amazing that a modern operating system like AntiX can still run on a system that is over 20 years old! So, if you have an old Windows computer sitting in your basement collecting dust, you could breathe new life into it as a Linux machine using a lightweight distro like AntiX.

The other type of system requirements is recommended requirements. Recommended system requirements list the hardware that you should have to maximize the system's potential. This

allows you to use applications such as video editing software or play 3D games, which require higher frame rates for smooth gameplay. When you look at the recommended requirements, you will notice that most distributions will now recommend a video graphics card and much higher processing and memory than the minimum requirements. However, you still need to be careful that you do not use incompatible hardware because it may be more powerful than the minimum requirements but may render the whole system unstable or unusable if you install an unsupported graphics card or other device.



When it comes to using Linux on a computer, it may already come pre-installed by the device manufacturer or you may need to manually install it. To manually install a Linux distribution, the installer is essential. It is usually a download on the distribution's website. For example, to install Ubuntu, open a web browser and go to ubuntu.com. The installer is under the download section. Several different versions of the installer may be present and choosing

which to download will depend on system requirements and user needs. For example, there will be 32-bit and 64-bit versions that depend on hardware capability, and there might also be long-term support and non-long-term versions. The file size will also depend on the installer's version.

The installer is usually in the ISO file format. It is a disk image for a CD or DVD. There are several ways of managing this type of file format, and the traditional way is to burn the disk image onto a CD or DVD. For the Ubuntu ISO file, Windows 7, 8, and 10 users will simply need the downloaded Ubuntu ISO file, a blank DVD, and access to a computer with a DVD burner. Right-clicking on the ISO file will show the context menu options, which includes "Burn disc image." Clicking it brings up the Windows Disc Image Burner dialogue, where we can select the disc burner, insert the blank DVD, click the burn button, and start the process. Older Windows versions, such as Windows XP and Vista, may need third party software, such as Infra Recorder, to manage burning ISO disc image files. Mac users will also go through something similar, and tutorials for both Windows and Mac image burning are on the Ubuntu website.

Another way to manage ISO files for newer systems, which can boot off USB thumb drives, is to create a bootable USB stick. Optical drives are becoming a thing of the past, as they are prone to physical damage. Bootable USB installers are now the preferred method for installing operating systems like Windows and Linux. There are also advantages to creating a Ubuntu bootable USB stick; for instance, it allows us to test out Ubuntu without touching the host computer's configuration. We can also carry this USB stick around and use it to boot Ubuntu on other computers, such as a friend's or at an internet cafe.

Creating a Ubuntu bootable USB still requires the ISO file, but, instead of going through the disc burning process, we need a third-party software such as Rufus, an open source software for creating bootable USBs. A sufficiently capable USB thumb drive is also necessary. For newer versions of Ubuntu, we need a 4GB or larger USB stick or flash drive. It should be empty, as it may require reformatting, which destroys existing files. The process is straightforward, and there are easy to follow prompts. It will also download additional files, so we need internet connectivity. A prompt asking for the write mode will appear and selecting ISO

mode at this point is recommended. It may take some time for the process to finish, and, once it is done, the USB drive is now bootable, and we can use it to install Ubuntu on a machine capable of booting through USB.

Before going through an actual install by taking over an entire system or booting alongside an existing operating system, we can test Ubuntu on a virtual machine, which is like another computer running inside your computer. A popular virtual machine software is VirtualBox. Just like downloading the Ubuntu ISO file from the Ubuntu website, we can download the VirtualBox from its website, virtualbox.org. Download the version that matches the operating system that VirtualBox will be installed on. Going through the VirtualBox installer should be straightforward, as it recommends installation defaults. Create a new virtual machine by naming it, setting a virtual memory (RAM) value, and creating a virtual hard drive disk (HDD). We can then load the downloaded Ubuntu ISO image as a virtual optical drive from which the virtual machine can boot. Starting this virtual machine is like turning on a physical computer, and it will boot from the loaded Ubuntu ISO file as if it were a DVD inserted in a DVD drive.

The Ubuntu installer starts with a language selection and presents two options: try or install. Selecting try loads Ubuntu as a live operating system, running temporarily as if it were installed, but it is not retained in the system, especially when the computer shuts down or restarts. Selecting the install option installs Ubuntu onto the system, whether it is an actual or virtual machine. The next prompt in the installation is the keyboard layout selection. After that, the installation presents options for a normal or minimal install. A normal install has all the preloaded software, while the minimal install has less. There will also be options to download updates while installing Ubuntu, and we can choose whether to also download some third-party software, some of which is proprietary. The next step presents options to either allow the system to be used exclusively for Ubuntu or to set HDD partitions so that we can boot Ubuntu side-by-side with other operating systems. The latter option is also called a multi-boot system. Installing a multi-boot system is more complicated, and there are a ton of resources online, especially from Ubuntu's website and online community. The next major step in the process is supplying user credentials: the user's, computer's name, username, and password. After supplying user

credentials and selecting whether the user is logged in automatically or not, the system installation starts.

The first boot screen of Ubuntu shows the "What's new in Ubuntu" dialogue, which is like a walkthrough of some of the operating system's features. The next section of the first boot shows Livepatch and is where we can create a Ubuntu single sign-on account. The next section asks whether the user wants to send information to Ubuntu to better help them in the OS's development. After going through the first boot screen sections, the Software Updater may run if it detects updates on some of the system software or applications. Always keep them up to date.

Operating System Differences

There are hundreds of operating systems in use today. These range from commonly used ones to those that are specifically developed for unique systems. Most readers are familiar with the two most popular operating systems in the world: Windows and macOS.

Microsoft Windows dominates the world as the operating system of choice, with nearly 90% of all desktop and laptop computers running some version. Many home users and companies prefer Windows for their everyday use, productivity needs, and gaming. This is also driven by the fact that most software developers release their products on the Windows platform due to its widespread popularity, creating a self-perpetuating feedback cycle that continually increases Windows popularity.

The second largest operating system is Apple's macOS. With the increased popularity of the iMac and MacBook computer lines in recent years, macOS now makes up about 9% of all desktop and laptop operating systems. While slowly gaining market share, macOS still cannot take away large portions of Windows's dominance since macOS only runs on Apple's own computer hardware.

In third place, the Linux operating system is found on less than 2% of all desktops and laptops. While this is a small amount, remember this doesn't mean Linux isn't important. These numbers are for home and business desktop users. When it comes to servers, Linux dominates Windows and other operating systems with over 75% of the market share.

For home users, the overall user experience of Windows and

macOS made them both very popular. Linux, on the other hand, is known as the desktop OS for computer experts or hackers. People perceive Linux as difficult to use, but that simply isn't true, as you will see for yourself throughout this book.

Windows is also the most prone to malware attacks because of its larger user base. This makes it susceptible to attackers who circumvent security measures to steal data for their own financial gain. After all, if you were a criminal and spent the time to create a piece of malware, wouldn't you create it for the operating system that nearly 90% of the world uses? Of course, you would—and so would they!

Unlike for Windows, malware is very unlikely to affect Linux since the source code is openly available, and the community of users can always modify it when they detect bugs or vulnerabilities. Like Linux, macOS has its roots in a Unix-like operating system, and therefore also enjoys lesser exposure to vulnerabilities than Windows. With that said, macOS is still more susceptible to malware than Linux due to its larger user base.

Windows is a closed source, proprietary software product. Using Windows requires a valid license, which can be purchased for a nominal fee of several hundred dollars. Linux, on the other hand, is open source and, therefore, free to download and use. The third option, macOS, is sort of a hybrid of these two models. The software is licensed under a free to use model (like Linux), but it is proprietary software (like Windows). Also, macOS can only run on systems built by Apple, as per their licensing agreement. This means that to use the macOS, you must purchase the hardware from the tech giant with higher prices.

Unlike Windows or macOS, though, you can run Linux as an operating system from only the command line. This is especially helpful when you need an OS to run on systems with limited resources, such as old computers or for specific purposes, such as servers and embedded systems. We will further discuss how to use the Linux command line later in this book, spending a good bit of time covering the different commands in detail to help you become proficient at it.

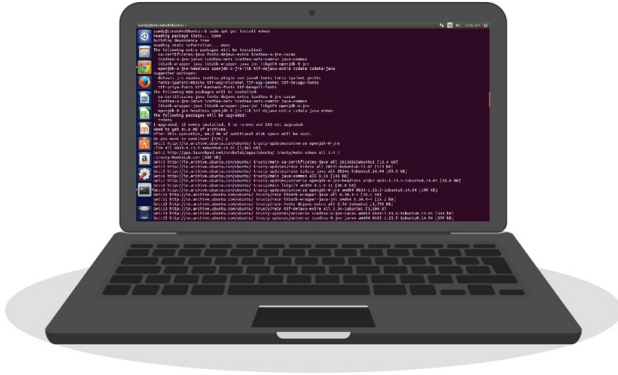
Windows and macOS run entirely on graphical user interfaces or GUI. Though Microsoft did develop a command line only operating system called MS-DOS in the early 1980's, with the steady development of GUI-based OS starting from the mid 80's, MS-DOS

slowly faded into merely a command prompt inside Windows, and it is not a complete operating system into itself. For the macOS system, there is a command line environment known as the terminal, but most users never use it and instead rely solely on the graphical user interface.

Overall, each operating system has its own pros and cons. I will leave it up to you to decide which operating system you prefer for your own desktop environment. However, since this is a Linux course, we will focus the rest of this book on its operations and usage. We will, however, compare the various Linux concepts to Windows from time to time in order to help you better relate these new concepts to things you might already know.

CHAPTER THREE

Open Source Software



OBJECTIVES

- Understand the concepts of open source software
- Explain the costs of open source software
- List basic desktop and server applications
- List different programming languages and their tools
- Explain how package installation and repositories work

Open source software is software released under a license in which the copyright holder grants users the rights to study, change, and distribute the software to anyone and for any purpose. As such, the source code (the human readable core of the software) is available for inspection, modification, and enhancement.

Open source software is usually developed in a collaborative public manner and is an example of how open collaboration can create some exceptional products and services.

In the early days of computing, programmers and developers shared software in order to learn from each other and to evolve the computing field. Eventually, the open source notion moved to the wayside, as many developers opted to commercialize their software in the 1970s and 1980s. Still, many academics and some developers still create software collaboratively, even to this day.

Because of this collaboration, software developers may want to

publish their software with an open source license, such as the GNU General Public License (GPL), so that anybody may also develop the same software or understand its internal functioning. With open source software, anyone can modify it, port it to new operating systems and instruction set architectures, share it with others, or even market it in some cases. According to some scholars, there are several policy-based reasons for the adoption of open source, including the heightened value proposition created by the open source movement. For example, many open source software products have greater security, affordability, transparency, interoperability, scalability, and localization support than commercial variants.

The Open Source Initiative is a public benefit corporation that promotes the usage of open source software. They published the Open Source Definition (OSD), a document that determines whether a software license can be labeled with the open source certification mark. This mark serves as a distinctive label to show that a software's attributes meet the requirements of the OSD.

Cost of Open Source Software

Open source software is generally freely available for download, so the cost to acquire a copy is very low. But like any software project, whether open source or proprietary, costs can start to pile up when it comes to technical support, training, and administration or maintenance.

In the early days of computing, almost all software was produced by academics' and corporate researchers' collaboration. They often shared software as public-domain. This meant that there was absolutely no ownership, such as copyright, trademark, or patent. As such, they generally distributed software under the principles of openness and cooperation. It was simply not seen as a commodity that could be sold for profit. Such communal behavior later became a central element of the so-called hacking culture, which is a term with a positive connotation among open source programmers.

During this early period, the source code was generally distributed with the software machine code. This allowed users to modify the software themselves if they needed to, since the code would often not run on a different hardware setup or operating system without modification. This also allowed any user to fix bugs

or to add new features, if they so desired.

Long after the invention of computers, industries began widespread adoption of computing power and its advancing technologies. With this widespread adoption, a need arose for new hardware and software to be developed to meet this increased demand. During this time, hardware manufacturers would create most software as well, and this dramatically drove up the software's cost. To compete with the hardware manufacturers who bundled software with hardware, a brand-new software industry began to grow. To increase revenues and keep up with rising costs in software development, a general trend began of no longer openly distributing source code, but instead making it proprietary and maintaining a legal copyright on the programs created. Now, to legally run the software, an individual or a company would need to acquire a license, resulting in profit for the software developers.

However, a resurgence in research and alternatives to lower the cost of using and maintaining computer systems once again resurrected the idea of free open source software in the late 20th century. Open source software developers now had to look for ways to continue developing software while allowing people to use it for free. Some of the developers resorted to private funding, crowdfunding, and even accepting donations from users to keep their projects alive. Since donations are voluntary, this still allows other nondonating users to use the software for free.

Others moved to a freemium model, often striking partnerships with other companies to show advertisements or install other software from within their software. This allows their software development to be funded while still allowing end users to use their software freely.

Still others chose to adopt a freely distributed, open source model that relies on support agreements put into place by the end user. This means you can use the software for free. but if you want support from the developer when you have a problem, then you must pay a service fee. This model was popularized by Red Hat Enterprise Linux, which is completely free to use, but charges for official support, training, and certifications.

Desktop and Server Applications

Scan QR code to watch a video for this topic



Most desktop applications that come pre-installed or that can be later installed on Ubuntu are free and open source alternatives to more popular commercial software. However, it does not mean that they lack performance. In fact, most of these programs are now mainstream, and they are sometimes even preferred over commercial software. Many of these free and open source applications function similarly to their commercial counterparts with slightly subtle differences.

LibreOffice is a free and open source office suite, found pre-installed in most Linux distributions like Ubuntu. It is like Microsoft Office. Included in the LibreOffice suite is Writer, a word processing program like MS Word. It allows us to write notes and documents that we can also format according to preferences and use. Also included in this office suite is Calc, a spreadsheet program similar to MS Excel or macOS's Numbers. For creating presentations such as those on MS PowerPoint or Keynote, Impress is the program.

Moreover, VLC media player is a free and open source,

portable, cross-platform media player software and streaming media server available in Ubuntu. The good thing about this program is that it is cross-platform, which means that it can run on other operating systems, such as Windows, macOS, and even on mobile phones. Its main advantage over other media players is its ability to play almost any audio and video format. and it supports a wide range of codecs.

A program to manipulate images on Ubuntu that's like Adobe Photoshop is GIMP. An acronym for GNU Image Manipulation Program, GIMP is a free and open-source raster graphics editor for image retouching and editing, free-form drawing, converting between different image formats, and more specialized imaging tasks.

For those looking for video editing programs like Adobe Premier or Final Cut Pro, Kdenlive is a free and open source alternative. For working on audio files, the program to use is Audacity: a powerful, cross-platform, and open source digital audio editing and recording application software.

The internet browser that comes pre-installed in Ubuntu is Firefox, although users can always add others such as Google Chrome, Opera, or many others.

For server and network administrators, it is also important to have applications that extend Ubuntu's capabilities beyond desktop tasks. One of these server applications is Wireshark. It is a free and open source packet analyzer. Admins use it for network troubleshooting, analysis, and software and communications protocol development.

GParted is a graphical user interface for managing disk partitions. It can create, format, and resize partitions without needing to memorize the code entered to do those functions.

An important aspect of managing servers and networks is the ability to backup data to retrieve or restore from it in the event of system crashes or failure. An application called Timeshift can take scheduled snapshots of the system to give the ability to restore the system to a previous snapshot when there are accidental deletions or changes render the current system unstable.

To quickly create text for coding or system scripts rather than office documents, Atom is a good application. It is like Notepad on Windows.

Another helpful tool for system and network administrators is

PuTTY. PuTTY is a free and open source terminal emulator, serial console and network file transfer application. It supports several network protocols, including SCP, SSH, Telnet, rlogin, and raw socket connection.



One of the great things about Linux is that it has embedded support for different programming languages. Here are some of those languages and the tools that we can use to write, read, and execute them.

JavaScript is simply a text file with .js as its extension. We can embed this within HTML, or it can be its own file. It can be as simple as a script to show "Hello World" when opened in a browser, or it can have more complex functions, such as creating a list.

Python is a great scripting and interpreted language. There are two ways to write python code: in an interpreted environment or from inside a text file. For the interpreted environment, open a terminal and type python3 (3, since it is the latest version of Python). This will bring up a prompt that will take direct

commands. Inputting `print("Hello, World")` will return the text Hello, World right below that command line. We can achieve this by creating a text file, typing in the exact same command, and saving that file with `.py` extension. Back at the terminal, type `python3`, insert a space, and append the filename of the saved text file; it will show the same results.

PHP originally stood for Personal Home Page but now it stands for the recursive initialism, PHP: Hypertext Preprocessor. As we servers heavily use it, so does Linux, as it is an operating system primarily used for web servers. It is just like JavaScript, as it works with HTML files. As such, a PHP file indicated by `.php`, can run directly in a browser.

Another programming language, Java, is a great language because it also works on a lot of different operating systems, including Windows and Linux. This should not come as a surprise for most smartphone users running the Android OS, as all the applications on their phone are written in Java. Java is a compiled language. Looking at a `.java` file, the objects in it are just text that the computer will not understand unless it is converted into an executable binary. So, the terminal first compiles the `.java` file by executing the `javac` command followed by the filename with `.java` extension. Once it is compiled, run the object by typing `java` and whatever is the object's name is in the `.java` file. Java is a bit more complicated than JavaScript, Python, or PHP, but is extremely powerful because we can write full-fledged programs using this language alone.

CSS, also called "style sheets", is used with HTML pages; they describe the content and how things should be displayed within that page. A typical `.css` file contains page attributes, such as how the body of the page should look, what color the background should be, the type and size of font, page margins, etc. Modifying the style sheet easily changes the look and feel of a webpage and is easier than going back and modifying the whole HTML document.

C++ is a compiled language like Java and is a low-level programming language. It is a little bit more complicated, needing to compile the binary before executing it; but it is very powerful, as most of the big-name software like Adobe applications, Google applications, and even MS Office and older versions of Windows were all written in C++.

A newer object-oriented programming language is Go. Like Java

and C++ , it also needs to be compiled. Go is designed specifically as a systems programming language for large, distributed systems and highly scalable network servers. Since Go is faster to compile than C++ and is much easier to maintain than Java, it replaced them in Google's software stack.

C is a low-level programming language from which C++ evolved. In fact, the "hello, world" example, originally written in C, has become the model for writing introductory programs in most programming textbooks. The program prints "hello, world" to the standard output, which is usually a terminal or screen display.

In addition to C and C++ , another member of the family is C#. It is a general-purpose, multi-paradigm programming language. It is used from within a development environment such as MonoDevelop.

Ruby is a popular web development programming language. Ruby files are indicated by a .rb filename extension. Ruby is like Python, as it can run in an interpreted environment or from inside a text file.

There are many more development languages and programming tools available to Linux users. The ones mentioned here are just the more popular ones. Selecting which to use depends on the user's knowledge of the language as well as the language's fit for purpose.

Package Management and Repositories

There are many ways to install software on a computer powered by Linux. The most common method is to use a package manager and software repository. Since there are a wide variety of Linux distros, there are also a wide variety of package managers. Each Linux distribution compiles its own software with its desired library versions and compilation options. Linux applications generally don't run on every distribution, but instead need to be compiled for a particular distro. Even if there was a universal Linux binary, its installation would be hindered by the various competing package formats and their availability within a distro.

Therefore, if you locate the website of a Linux application, you will likely see a variety of download links for different package formats based on the distribution you are using. If the author of the application you want to install doesn't already have a package created for your distribution, then you may have to download the source code for the application and manually compile the binary for

the application yourself.

Unlike a Windows user, most Linux users don't normally download and install applications from the applications' websites because of the challenges described above. Instead, each Linux distribution usually hosts its own software repositories. These repositories contain software packages compiled for a specific Linux distro and version. For example, if you're using Ubuntu 18.04, the repositories you will use contain packages specially compiled for Ubuntu 18.04 and not for another version like 16.10 or 19.04.

These package managers automatically download the appropriate package from its configured software repositories, install it, and automatically set it up for the end user. This contrasts to installing software in Windows or macOS, where you must click through wizards or locate executable files on websites. When an update releases, the package manager notices its availability, downloads the appropriate update, and installs it for you.

On Windows and macOS, each application has its own software update program to receive an automatic update, but in Linux the package manager handles updates for every piece of software installed from the software repositories.

If you are a Windows user, you are probably used to seeing an .exe or .msi file as the installer files for applications. If you are a macOS user, then you probably are used to downloading a .dmg or .app file as the installer. Linux, though, uses several package formats for its installers.

Packages are essentially archives containing a list of files. The package manager opens the archive and installs the files to the location specified within the package. The package manager is responsible for maintaining a record of which files belong to which packages. The package manager also must remain aware of their version and their latest status. These package files can also contain scripts that run when the package is installed and removed.

The three most popular package file formats are .deb, .rpm, and .tar files.

- .deb is used in Debian Linux and Debian-based distros like Ubuntu, Kali Linux, and Tails
- .rpm is used by the Red Hat Package Manager and RPM-based distros like Fedora, openSUSE, and CentOS

- `.tar` (also `.tgz` or `.tar.gz`) is used a universal package format and is used in distros like ArchLinux and Slackware; `.tar` is an uncompressed archive and `.tar.gz` is a compressed archive

While Linux distributions ship with their own repositories pre-configured, you can also add other repositories to the system. You can then install software packages from that repository and receive any additional updates for them automatically.

For example, Ubuntu and some other Debian-based distributions offer a wide variety of personal package archives (PPAs), which contain software compiled by individuals and teams. These PPA repositories often get updates and beta versions of software before the official Ubuntu repository.

Remember, these PPAs are not officially endorsed by the distribution, so they may contain bugs, security vulnerabilities, and other nefarious things. For this reason, I recommend only using official repositories from your distribution for any production systems.

CHAPTER FOUR

The Desktop Environment



OBJECTIVES

- Compare and contrast various desktop environments
- List Linux alternatives to common software programs
- Compare and contrast various package managers

One of the more distinct advantages of Linux over Windows and macOS is its ability to let users have complete freedom of choice. Unfortunately, this is also one of the most confusing things about Linux, as that choice can become quite overwhelming.

For example, let's pretend that you choose Ubuntu from among the hundreds of Linux distributions in the marketplace. Ubuntu is quite popular, and it is a good choice since it has a large user community and excellent lifecycle support. Even though you chose Ubuntu, you still have some decisions to make before you can download and install it. This is because there are eight official versions that all look and behave differently, even though they are all considered Ubuntu. One of the major differences in these eight divergent versions of Ubuntu is the desktop environments.

To better understand what a desktop environment is, start with the core of any operating system, the kernel. Linux, Windows, and macOS all have a kernel, a specialized piece of software that directly controls the hardware. This kernel translates the commands

from a piece of higher-level software (like an application) into something the hardware can understand and act upon. The kernel is responsible for the intelligent management of hardware resources, including memory management, for various software and utilities.

The kernel is not a piece of software that requires a graphical interface, but instead it operates behind the scenes. This specialized software consumes the least amount of resources possible, and we can access it directly via the command line using embedded tools. To do certain tasks on the command line, a user needs to be familiar with text-based commands, what these commands do, and the syntax or rules needed to run these commands.

In fact, by using just the command line environment, you can perform nearly any action you wish on a system, including browsing the web, checking email, playing music, moving around the directories on the hard drive, and much more. Unfortunately, most users find the command line not very easy to use, so operating systems evolved into a more user friendly, graphical user interface (GUI) environment. In Windows and macOS, for example, you may never even access the command line or the kernel directly, but instead you would use the GUI for your desired actions. This GUI then translates your series of mouse clicks and keyboard inputs into the appropriate command line actions on your behalf. Basically, in Windows and macOS, only system administrators, programmers, and developers ever need to access the command line.

In Linux, on the other hand, it is often much quicker and easier to perform an action using the command line. For this reason, we will spend an entire chapter on properly using the command line. In this chapter, though, we will focus instead on Linux's various graphical user interfaces.

A desktop environment is the place where most user actions occur. This graphical user interface (GUI) contains a file system manager, shortcuts to all available applications, various menus to easily perform common tasks, and serves as your window into the operating system and its functions. Your desktop environment serves as your desk, where all your work and tools are easily within reach.

While Windows and Mac each have an embedded and inseparable desktop environment as part of their operating system, Linux uses a more modular approach to desktop environments. In fact, there are more than a dozen different desktop environments

that are officially supported in Linux. These environments can be mixed and matched according to the user's preferences or based on the specific purpose of the distribution itself.

Many Linux distributions come with variants featuring several different desktop environments. As previously stated, Ubuntu has eight different available versions and numerous different desktop environments represented among them. By default, Ubuntu comes with the Unity Desktop Environment, which has an attractive look and intuitive workflow. In addition to this default option, Ubuntu also has other “flavors” (different versions) that are each shipped with a unique desktop environment. The Kubuntu variant relies on the KDE Plasma desktop environment, which is slick, flashy, and highly customizable. The Xubuntu variant, on the other hand, uses the lightweight Xfce desktop environment, which is robust and can run on older computers with as little as 1 GB of memory and a 700 MHz processor.

Programs and Software

After installing a Linux variant and a desktop environment, a user can then run various programs and software to increase their productivity. Just like Windows and macOS, there are millions of available programs available. These programs cover a wide variety of functions. Because Linux is open source, much of the software available for Linux is also open source and, therefore, free to download and use.

If you are a Windows or macOS user, you probably are familiar with the Microsoft Office suite of products for workplace productivity. This includes word processing (Word), spreadsheet (Excel), presentation (PowerPoint), and database (Access) tools.

Since Microsoft Office isn't available on a Linux machine, most Linux users rely on LibreOffice (formerly OpenOffice) to fulfill these workplace functions. LibreOffice is a feature-rich toolset that includes word processing (Writer), spreadsheet (Calc), presentation (Impress), database (Base) tools, and many others. The LibreOffice suite is fully compatible with Microsoft Office, as well, allowing it to read and write to common workplace file formats like .doc, .docx, .xls, .xlsx, .ppt, .pptx, and more. Additionally, LibreOffice supports its own native format known as the Open Document Format (ODF), which is a modern and open standard for office productivity files. LibreOffice is also cross-platform compatible,

with versions of the software available for Linux, Windows, and macOS. To learn more about LibreOffice, or to try it yourself, visit <https://www.libreoffice.org>.

If you want to watch videos or play music on your Linux workstation, you can install one of the best media players available by downloading VLC. The VLC media player can play just about any media file you might throw at it, including damaged, incomplete, or even unfinished files. VLC is a truly free multimedia solution that works on every major operating system, including Linux, Windows, macOS, Android, and iOS. I have personally used VLC as my media player of choice on every computer I have ever owned for over 15 years, and it hasn't let me down yet. If you are looking for a great alternative to the Windows Media Player or the QuickTime player, you should head over to <https://www.videolan.org>.

If you want to browse the internet, you will need a web browser. Although Linux doesn't support proprietary web browsers like Microsoft's Edge or Apple's Safari browsers, it does support several cross-platform web browsers. This includes Mozilla Firefox, Google Chrome, and the Opera web browser. While all of these are free to use, only Mozilla's Firefox is open source software, since the Google Chrome and Opera browser's source code is not freely distributed.

Linux is also well-suited for editing photos, videos, or audio. If you need to retouch, and edit, free-form draw, convert between image formats, or perform specialized photo and image creation tasks, the GIMP (GNU Image Manipulation Program) is an excellent option. Whether you are a graphic designer, photographer, illustrator, or scientist, GIMP provides you with sophisticated tools to get your job done. GIMP is a free and open source raster graphics editor that operates much like Adobe Photoshop, only without the heavy price tag. While Adobe Photoshop is only available for Windows and macOS, GIMP is a cross-platform software for Linux, Windows, and macOS. GIMP has been around since 1995, and it has kept improving ever since. GIMP is a powerful image manipulation program, making it a suitable replacement for Adobe Photoshop for most users. To learn more about GIMP, or to try it yourself, visit <https://www.gimp.org>.

If you like to edit video, Linux has an excellent non-linear video editing software called Shotcut. This program supports video and audio editing using a timeline view of multiple tracks, much like Adobe Premiere or Apple's Final Cut software tools. In fact, Shotcut

is so powerful that it can even support editing 4K video footage. Like most Linux software, Shotcut is a free and open source application. Shotcut is also a cross-platform tool that can be installed on Linux, FreeBSD, Windows, and macOS. You can find out more about Shotcut at <https://www.shotcut.org>.

If you are thinking of becoming a rock star one day, you may find that you need to edit some audio for your demo CD. All jokes aside, though, Linux has tools for that too. Audacity is a free and open source digital audio editor that is available for Linux, Windows, and macOS. This program allows a user to record audio from multiple sources and conduct post-processing for all types of audio formats. Whether you want to mix an entire album or if you are thinking of starting a podcast, Audacity can handle it all. Audacity is comparable to the Adobe Audition or Apple's Logic Pro X software used by many professionals in the recording industry. You can find out more about Audacity at <https://www.audacity.com>.

It should be obvious by now that there are numerous tools available to perform just about any function you could imagine within Linux. The software mentioned above are just a small sampling of the options available. For the LPI Linux Essentials certification exam, you should be aware that proprietary software, such as Microsoft Office, is not available for Linux, but there are a wide variety of open source replacements available to replace them, such as LibreOffice.

Managing Software Packages

When we install a Linux distribution, it normally comes with many pre-installed applications so that a user can get to work right away. This may include programs such as office productivity suites, media players, and web browsers. In addition to these pre-installed applications, Linux distributions can also access package repositories that contain a vast collection of easily installable application. These are called packages.

A package is a compressed file archive containing all the files that come with a particular application. The files within the package are usually stored according to the relative installation paths within a distribution. Most packages also contain distribution-specific installation instructions and a list of any other packages that are prerequisites for installation. In Linux terminology, these prerequisites are called dependencies. Simply put, a dependency is a

broad software engineering term that refers to a piece of software that another software relies on for installation. For example, the VLC media player has a large list of dependencies that we first need to install, such as various video codecs to allow playback within the media player.

Linux packages utilize three common file types: `.deb`, `.rpm`, and `.tgz`. The filename extension of the package usually indicates which Linux distribution the software was compiled to work with.

For example, a `.deb` package contains software compiled and packaged for Debian and other Debian-derived distros like Ubuntu, Kali Linux, and Tails.

The `.rpm` package format was originally developed for Red Hat Linux, but it is also used in other distributions like Fedora and openSUSE.

The tarball or `.tgz` package format is sometimes referred to as the universal package format. This is because a tarball just takes multiple files and sub-directories and compresses them using the gzip compression software to save bandwidth for users trying to download and install them. If the tarball is uncompressed, it usually ends with `.tar` in its filename.

Since Linux packages do not usually contain the dependencies necessary to install them, many Linux distributions use package managers that automatically read dependencies files and download the necessary packages before proceeding with the installation. Some common package managers are APT, YUM, and Pacman. While the various Linux distributions offer roughly the same types of applications, there are still many different package management systems in use today.

For instance, Debian and Debian-based distributions use the `dpkg`, `apt-get`, and `apt` tools to install software packages using the `.deb` package format. `Dpkg` is the software at the base of the Debian package system and is a low-level package manager. Being a low-level package manager, `dpkg` can only install a package if all the dependencies are already installed. APT, the Advanced Package Tool, is a higher-level package management tool. Ubuntu created APT to include the `apt-get` and other tools. Since APT is a higher-level package management tool, it can find, install, upgrade, and remove packages and their dependencies for you.

If you are using a distro like Red Hat, Fedora, or CentOS, then you will download and use packages in the `.rpm` format. These

distros use the RPM, YUM, and DNF commands instead of dpkg and APT tools. RPM is both a package format and a low-level package management tool. YUM is a higher-level package manager, and DNF is a newer replacement for YUM. In fact, DNF stands for “dandified yum”.

Because application packaging is different for each distribution family, it is important to install packages from the repository that is designed for your distribution. Luckily, users usually do not have to worry about these details because the distribution’s package manager will choose the right packages, the required dependencies, and apply the necessary updates when they become available.

CHAPTER FIVE

The Linux Shell

```
root@diontraining:~/Dekstop$ ls
name.txt          number.txt      tar1.tar        text.txt.gz     zip4.zip
name.txt          number.txt~    test3           zip1.zip
numbers-long.txt  site.txt       test4           zip2.zip
numbers-long.txt~ site.txt~      text.txt~      zip3.zip
root@diontraining:~/Dekstop$ unzip -l zip4.zip
Archive:  zip4.zip
  Length      Date    Time    Name
  -----
  358854      2019-05-05  08:50   silver_drum.jpg
  385235      2019-05-05  08:50   gold_shoes.jpg
  -----
                    2 files
root@diontraining:~/Dekstop$ unzip zip4.zip
Archive:  zip4.zip
  inflating: silver_drum.jpg
  inflating: gold_shoes.jpg
root@diontraining:~/Dekstop$
```

OBJECTIVES

- Contrast the desktop environment to the shell
- Compare and contrast various Linux shells
- List basic shell commands and their usage
- Explain command line syntax
- Understand variables and their use in the shell
- Explain why quoting is used in the shell

While the desktop environment is fully based on a graphical user interface, the Linux shell is based on a text-based, command line environment. At this point in your Linux journey, it is important for you to understand at least the basics of the shell since it performs all the commands in the background on behalf of the desktop environment.

Simply put, the Linux shell is the program that takes commands from the keyboard and presents them to the kernel for execution. In the old days, most operating systems were only available as a text-based, command line interface. These days, however, the desktop environment or GUI sits on top of the command line interface (CLI) and makes the execution of commands and task much easier for the end user.

When the user clicks on a shortcut located on their desktop, that

shortcut may contain a predefined command bundled together with the correct syntax to execute a program. For example, if you click on the shortcut for Google Chrome on your desktop, it may be configured to open the web browser and load your homepage. For this to occur, you simply double clicked the shortcut, which caused the GUI to translate that action into a series of commands within a hidden CLI.

The Linux command line, or shell as it is more properly called, can be launched or started in a GUI window known as a terminal program. Some Linux distributions, such as ones designed for server use, will eliminate the desktop environment altogether to save on processing and memory resource. This allows a user to directly log into the system locally via a text-mode console or to connect to a shell when they remotely log into a server using a text-mode login protocol, such as Telnet or SSH.

Types of Shells

Most Linux distributions systems have the option of using several different shells, though. These include the Bourne Again Shell (bash), Korn shell (ksh), the TENEX C Shell (tcsh), and the Z shell (zsh).

The most common shell is known as bash. It is based on an older shell called the Bourne Shell. Bash is a program that operates within the command line interface to read and execute commands from the command line or from a file known as a shell script.

The Korn shell is named after its creator, David Korn, and served as the basis of the POSIX.2 operating system. Tcsh is a shell that improved upon an earlier shell known as csh (C shell) by adding command line completion, command line editing, and some other additional features. The csh is the default shell in BSD-based operating systems like FreeBSD. The zsh is an interactive login shell and command line interpreter that was selected as the default shell within macOS Catalina (10.15) and newer macOS versions as a replacement to bash.

For any demonstrations within this book, we will rely on bash since it is the default used by Ubuntu and is commonly used across the Linux community. Most of the other shells listed above are also compatible with Bash and provide a similar functionality, albeit with some subtle differences.

Each Linux user account specifies its own default shell, so

individual users can change their shells if they like. This can be done with account management tools, such as `usermod`, which we will describe in detail in a later chapter.

If you are using a desktop environment, then you can access the shell by running a program known as a terminal emulator (often simply called the terminal). The terminal opens a window and allows the user to interact with the shell from within the desktop environment.

Typically, a desktop environment has its own terminal, so the terminal program you will use may depend on the desktop environments you have installed. Many terminal programs include the word `terminal` in their names, such as the `gnome-terminal`. Others, like the K Desktop Environment's (KDE) `Konsole` and the generic `XTerm` do not specify which desktop environment they must be used with. Again, Linux provides users with numerous terminal emulators to choose from, including `rxvt`, `kvt`, `nxterm`, and `eterm`.

While there are many different terminal emulators, they all do basically the same thing. They provide an end user with direct access to an interactive shell session in which they can issue commands to the operating system.

The exact method of launching a terminal differs from one desktop environment to another. Normally, an entry for the terminal can be found in the desktop environment's menus. Many desktop environments also provide a search function where you can find a full list of all the different terminals installed within that system.

For example, in Ubuntu's Unity desktop, clicking the Dash icon (the Ubuntu symbol) located at the top of the application launcher will show a search bar. If you begin to type the word "terminal," it will show the available terminal applications within its search results.

A quicker way to launch the terminal is to use the keyboard shortcut by simply pressing `CTRL + ALT + T`.

Most terminal programs support tabs, which are like the tabs you use within a web browser. It can be helpful to have multiple tabs open because it enables a user to run multiple programs simultaneously, work easily in multiple directories, or run programs both as a normal user and as root. On a Linux system, the root user serves as a super user or administrator.

Basic Commands

To utilize the shell, you need to be able to use some basic commands. The first ten commands any Linux user should learn in the shell are `ls`, `cd`, `mv`, `mkdir`, `touch`, `rmdir`, `rm`, `locate`, `clear`, and `man`.

The `ls` (list) command is used to display the files, folders, and directory information within the shell. This is the equivalent of looking at the list of files/folders within a folder in a desktop environment.

The `cd` (change directory) command allows a user to switch between two directories. This is the equivalent of double clicking a folder within a desktop environment.

The `mv` (move) command allows a user to move a file/folder to another folder/directory using the shell. This is the equivalent of a user dragging a file/folder into another folder within a desktop environment.

The `mkdir` (make directory) command is used to make a new empty directory within the file system. This would be the equivalent of a user right-clicking and selecting “New -> Folder” within a desktop environment.

The `touch` command is used to create or make a new, blank file using the command line. This would be the equivalent of a user right-clicking and selecting “New -> Text Document” within a desktop environment.

The `rmdir` (remove directory) command is used to delete an empty directory within the file system. This would be the equivalent of a user dragging a folder to the recycle bin or the trash can within a desktop environment.

The `rm` (remove) command is used to delete a file or directory within the file system. Since the `rmdir` command will only delete an empty directory, the `rm` command will remove both files and non-empty directories. This would be the equivalent of a user dragging the contents of a folder to the recycle bin or the trash can within a desktop environment.

The `locate` command finds a file within the file system. If you don't remember where a certain file was saved and stored in your file system, the `locate` command can really be useful. This would be the equivalent of using the search or spotlight feature in other desktop environments.

The `clear` (clear the screen) command erases the contents of the terminal display. This is useful when there is a lot of information

and previous commands remaining on the screen are cluttering the terminal with extraneous information. With the clear command, the user will be brought to the top of the terminal's display and returned to the shell prompt and the current working directory. This would be the equivalent of closing out numerous open windows in your desktop environment so that you can refocus your attention on the one window you need to work within.

The man (manual) command shows all the information about a given command. When a user is unsure of how to perform an action or how to properly input the syntax to a command, the man command is essential. This is the equivalent of a user opening the help file or a document containing all the details about how a command works.

Command Line Syntax

When a computer is attempting to an operation (no matter which operating system it is running on), it can be summarized in just four steps:

1. The computer waits for user input
2. The user chooses a command to use
3. The user enters it via the keyboard or mouse
4. The computer executes the command

In a Linux, the shell displays a prompt where commands can be entered. This prompt usually consists of a user and host (computer/server) name, the current directory, and a final character. If you are using bash, that final character is the dollar sign.

For example, a terminal using bash is displayed below:

```
Last login: Sun Sep  8 09:00:30 on console
DionTraining:Documents jason$ $
```

The hostname is DionTraining, which represents the server I am logged into. The directory is the Documents directory, and the user is Jason. Last, we see the final character, which is the \$ since I am using bash.

To perform an operation, you must enter a command. A command is essentially a sequence of characters in a line which ends by pressing the enter key and is then evaluated by the shell. Many commands are inspired by the English language and form part of a dedicated command syntax language.

Commands must follow certain rules known as syntax for the shell to be able to interpret them properly. Just as our English language has proper grammar rules so that everyone understands what we are trying to say, so does the command language. For example, I wouldn't tell a server at a bar, "Some water I want," because it would confuse them. Instead, I would say "I want some water" because this is the natural order, or syntax, used in the English language.

To interpret a command line, the shell first tries to divide the line into individual words. Just like in the English language, words are separated by spaces. The first word on a command line is usually the command itself, while all other words on the line are parameters that explain what the command should do in more detail.

There are two main types of command parameters: options and arguments.

Options are parameters that start with a dash or tack symbol (-). These are usually referred to as switches. Each switch allows certain aspects of the command to be turned on or off. These options can be passed separately to the command by using a space and a dash to separate them, or you can combine several options behind a single dash. For example, the switches -l -a would also be equivalently written as -la.

Some programs have more options than can be conveniently mapped to single characters, since there are only 36 characters available (a-z, 0-9). To allow for more options, these programs will often support what is known as long options. Long options often start with two dashes and allow for better readability. For example, it is clearer that --all indicates the command does something to all the files than if the switch -a was instead used. You cannot combine multiple long options; for example, --all and --more cannot be combined into

--allmore.

Arguments are the second type of parameters used with commands. Arguments are any words that are not preceded by a leading dash. Most often, parameters are used to input the names of files or directories against which a command should perform its intended function.

For example, let's assume you want to list (ls) all the files located in the LinuxEssentials directory located under the

Documents directory. If I simply typed “ls LinuxEssentials,” I would receive a list with two file names: Questions.txt and StudyGuide.txt. Since I only used a command (ls) and argument (LinuxEssentials), I only received the least default information displayed (the names of the files in the directory).

```
DionTraining:Documents jason$ ls LinuxEssentials
Questions.txt StudyGuide.txt
DionTraining:Documents Jason$
```

If instead, I wanted to display the file names, the date created, user and group ownership, and the permissions for the files, I would have to turn on that option using a switch. This can be done using the -l switch, which turns on the ability for the ls command to display the long format view with this additional information.

```
DionTraining:Documents jason$ ls -l LinuxEssentials
total 16
-rw-r--r--@ 1 jason staff 4 Sep 8 10:07 Questions.txt
-rw-r--r--@ 1 jason staff 4 Sep 8 10:07 StudyGuide.txt
DionTraining:Documents Jason$
```

Unfortunately, this display is incomplete because I cannot see any of the hidden directories in the LinuxEssentials directory. To enable the ability to view these hidden directories, we need to use the -a option:

```
DionTraining:Documents jason$ ls -a LinuxEssentials
.  ..  Questions.txt StudyGuide.txt
DionTraining:Documents Jason$
```

Unfortunately, since we only used the -a option, we no longer have all the additional details from the long format displayed. But, if we combine the -l and -a options into a single option of -la, as shown:

```
DionTraining:Documents jason$ ls -la LinuxEssentials
total 16
drwxr-xr-x 4 jason staff 128 Sep 8 10:08 .
drwx-----+ 22 jason staff 704 Sep 8 10:06 ..
-rw-r--r--@ 1 jason staff 4 Sep 8 10:07 Questions.txt
-rw-r--r--@ 1 jason staff 4 Sep 8 10:07 StudyGuide.txt
DionTraining:Documents Jason$
```

The above examples are not designed to make you an expert in the ls command, but instead you should now see the concept of how we can put together the proper syntax to pass different options and arguments to a command. Notice that in all these examples, we passed the name of the directory whose contents we wanted to list (LinuxEssentials), as well.

We can summarize this generalized command structure as “command option arguments.” Each of these three portions answers a different question for the program’s execution:

~~What~~ **What** do?

~~Options~~ **Options** do it?

~~Arguments~~ **Arguments** it to or with?

While many programs use the generalized form of command options argument, not all commands insist on this. Some programs will allow a user to mix options and arguments arbitrarily and simply behave as if all options came immediately after the command. With commands, though, options are only acted upon by the program if they are encountered while the command line is processed in sequence from left to right. To avoid problems, I recommend that you stick with the command options argument format, unless there is a good reason to deviate from it.

Now, just like in the English language, if you don’t use proper syntax, then your recipient (the system) won’t have a good understanding of what to do. Some programs will simply stop and do nothing. Others may stop and provide an error. Still others may attempt to process the command, but severe problems can arise if the incorrect options or arguments were used.

Shell Scripting

Since there are thousands of commands available for the command line, it can be a monumental task for a user to remember all of them. In addition to that, having to sit behind the keyboard and type out each command can become time consuming as well. But, the real power of computing comes with the ability to simplify repetitive and time-consuming tasks for the user. To get it to do that, we can harness the shell’s power to automate things by writing shell scripts that contain all the exact commands we want to run.

In the simplest terms, a shell script is a file containing a series of commands. The shell reads this file and carries out the commands as if entered directly on the command line.

For example, I have a script that I use to collect evidence from a victimized computer during an incident response. Instead of having to type out all 10-15 commands myself, I simply run the script and it performs each of those commands listed in my script one at a time until they are all completed. Scripts, though, can be much

more complex. For example, you can add logic to them by using basic programming so that they can detect changes to the system based on events or even the time of day.

Variables

Another essential feature of using the command line interface is the ability to use a name or label to refer to some other quantity, such as a value or a command. This is commonly referred to as a variable.

At the very least, variables make your scripts and programming more readable for an end user. However, we can also use variables in more advanced programming when a user is in a situation in which the actual values aren't known before the program executes. Essentially, a variable serves as a placeholder that is resolved at the actual execution of the script.

A variable is an area of memory that can be used to store information and is then referred to by a name. Whenever the shell sees a word that begins with a \$, it treats it as a variable and attempts to determine what was assigned to it in order to substitute the real value in its place.

To create a variable, a user should enter the name of the variable followed immediately by an equal sign (=) within the script. Please note, spaces cannot be used when entering the name of a variable. After the equal sign, we write the information that needs to be stored. This value is then assigned to the variable whenever that line is executed within the script.

For example, `$month=May` would assign a value of “May” to the variable `$month`. Whenever the variable name (`$month`) is written in the script after this point, it will be substituted at the time of execution with the word “May”.

When assigning a name to a variable, there are three rules that must be followed:

- Variable names must start with a letter
- Variable names must not contain embedded spaces
- Punctuation marks are not allowed
-

To ease readability, most programmers will use the underscore character (`_`) in place of a space within variable names, since spaces

are prohibited. For example, if I wanted a variable to store information about which day of the week it was, I might name the variable as `$day_of_week`.

When a user starts a shell session, there are already some variables declared and configured by a startup script runs while loading the operating system. To see all the variables that are in the user's environment, users can enter the `printenv` command. A script will always include the name of the machine being used. Consider the following example of the output of the `printenv` command:

```
DionTraining:Documents jason$ printenv
TERM_PROGRAM = konsole
SHELL = /bin/bash
TERM = xterm-256color
PATH = /usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
PWD = /home/jason/Documents
LANG = en_US.UTF-8
XPC_FLAGS = 0x0
XPC_SERVICE_NAME = 0
SHLVL = 1
HOME = /home/jason
LOGNAME = konsole
_ = /usr/bin/printenv
OLDPWD = /home/jason
DionTraining:Documents Jason$
```

As you review the output above, notice that all environment variable names are uppercase by standard convention.

Quoting

As previously stated, the shell reads input from the user; parses that input into commands, options, and arguments; and then executes those commands. As each command executes, its command line options and arguments execute. Unfortunately, the shell treats certain characters in a special way when it encounters them in the command line. These special characters are called shell metacharacters. There are several shell metacharacters, including the space, dollar sign, star,

semi-colon, greater than symbol, question mark, ampersand, and pipe

(, \$, *, ;, >, ?, &, |).

Up to this point, you have already seen a few of these shell metacharacters. For example, the most common shell metacharacter is the space character that the shell uses to separate the command,

options, and arguments. Because of the space character's special status, the shell will not pass any spaces to the command. Instead, the shell uses this metacharacter to separate and identify individual command line arguments. As far as the shell is concerned, a single space or 100 spaces mean the exact same thing: "I am about to find the next argument to parse".

So, how can we pass a file or directory name as an argument that has a space character in its name? For example, we used "LinuxEssentials" as the directory name in the ls command example, but what would we do if the directory was called "Linux Essentials" instead?

To pass a metacharacter to the shell and have it treated as a normal character instead, we use quoting. Quoting prevents the shell from acting on and expanding the metacharacters. It causes the shell to ignore the special meaning of the character, so that the character gets passed unchanged to a command as part of an argument. Quoting uses either double quotes (" "), single quotes (' '), or backslash (\) characters.

When we use quoting, the shell will ignore any special meaning that a metacharacter has and treat it simply as plaintext. For example, a quoted space character will not separate arguments while an unquoted space character will. Normally, we use a semicolon to enter two commands on a single command line, but a quoted semicolon (";", ';;', or \;) will not perform this function.

While technically quoting is only applied to the individual shell metacharacters protected from the shell and not to the whole command line argument, you can also surround the whole argument with matching quote characters (for example, ls -la "Linux Essentials" instead of ls -la Linux" "Essentials). We may also use backslashes to quote or turn off the special meaning of each immediately following an individual character (for example, ls -la Linux\ Essentials). The shell will execute all three of these examples the same way and provide the exact same results.

```
DionTraining:Documents jason$ ls -la "Linux Essentials"
DionTraining:Documents jason$ ls -la Linux" "Essentials
DionTraining:Documents jason$ ls -la Linux\ Essentials
DionTraining:Documents Jason$
```

Quotes and backslashes simply tell the shell which parts of the input to treat as ordinary (not special) characters. Quoting only delimits or identifies a string of characters. The shell removes the

actual quoting mechanism (double quotes, single quotes, or the backslash) and does not treat it as part of the string passed to the command.



There are a couple of ways to access the Linux shell. One way is by right-clicking on the desktop and selecting Open Terminal in the menu. Another way is by clicking on the Applications menu on the taskbar and searching for Terminal. There may be several terminals present in the system, so the easiest way is the right-click option.

In distributions like Ubuntu, right-clicking on the desktop and selecting Open Terminal in the menu opens the default terminal. Inside the shell, you can then start issuing commands. But this necessitates using the proper syntax.

For example, the `ls` (list) command is similar to a directory listing or the `dir` command in Windows. Typing `ls` at the prompt will show the folders and files of the current directory. Usually, when opening the default terminal, the home directory is indicated by a `~` before the `$` sign. To change to a folder within the current

directory, we use the `cd`, or change directory command. Sometimes, the shell user can get disoriented because there are distributions that do not show which directory is being currently accessed. In Ubuntu, the shell user can keep track of which folder is currently accessed by looking at the directory structure before the `$` sign. But for other distributions where this is not present, simply typing the `pwd` command, for present working directory, will show the full path from the root all the way to the current folder.

To quickly create files within folders, the `touch` command will create an empty file of whatever filename it is given. If we go back to the commands section, remember that there are commands, options, and arguments. Commands tell the system what to do, options state how to do those commands, and arguments tell the system what to do the command to or with. With the `touch` command, there is no need to enter an option. Simply typing `touch test1.txt` creates an empty file named `test1.txt` in the current directory.

Going back to the `ls` command, we can use it with several options to do more than just list files and folders in the current directory. You may want to see more details and additional information about the files and folders `ls` lists. Using the `-la` option or long view with all the attributes, `ls -la` lists out files and folders with additional details such as permissions, ownership, and files size.

Previously discussed were variables and variable names with emphasis on the `printenv` command to print out environment variables. The last line of the output of this command shows the `OLDPWD`, which was the last directory accessed before being in the current directory.

Quoting is also important in command syntax because when improperly used the output is not achieved. For example, the semi-colon (`;`) normally runs two commands simultaneously. For example, by typing `echo Hello; ls`, we first print `Hello` and then list the files and folders in the current directory. To change the function of the semi-colon in the command line, the text that should contain semi-colon (`;`) as a character is placed inside quotation marks or we use a backslash (`\`) before the semi-colon to make the shell treat the semi-colon as regular text.

The Linux shell is powerful and useful. However, with more than a thousand individual commands and even more syntax options and arguments, it is virtually impossible to remember every single one of them. Fortunately, there are two easy to use resources that can provide you with a lot of information when you are using the command line environment. These resources are the `man` and `info` commands.

The `man` command contains the manual (man pages) for every command available on the system. This includes detailed information on what the command does, the specifics of how to execute the command, and what command line arguments or syntax they accept. The man pages are invoked by running a terminal and typing the command `man` followed by the name of the command you want information on. For example, if you wanted information on the `ls` command, simply type “`man ls`” and press enter to enter the man pages for the list command.

When you are reading the man pages, it is also possible to do a keyword search for specific information you are looking to find. If you know what you want to achieve with a command, but you can't remember the exact command or syntax, using this search feature is quite useful. To perform a keyword search across every man page on the system, you simply type “`man -k`” and indicate the search term. Do note that you must use the proper spaces between the command, option, and search term. If you are searching for a complex phrase of multiple words, don't forget to use quoting around the search term to nullify the special meaning of the space metacharacter.

If you already know the command you want to use, you can also search within a single man page. To do this, press forward slash (/) followed by the term to search for and hit 'enter'. If the term appears multiple times, cycle through them by pressing the 'n' button for each additional entry.

It is important to know which command line options we can use to modify the behavior of a command to suit our needs. Using the man page, you can read all about the different options supported by a command. For example, if you read the man page for the `ls` command, you will see that there is a longhand and shorthand version of the command's options.

For example, I previously used the `ls` command with the `-a` option to list all the directories (including the hidden ones) within a

directory. The `-a` is the shorthand version of this option, but there is also a longhand version (`--all`) that we can use for the same functionality. This longhand version is just a more human readable form, but we can use either since they both do the same thing.

There are options to use the longhand or shorthand versions, though. By using the longhand version, it can be easier for users to remember what the command does. By using the shorthand version, though, a user can combine multiple options together more easily (for example, `-l` and `-a` become `-la`). The shorthand version is also quicker and easier to type out.

Another command that will display documentation and information to the user is the `info` command. To use this command, simply type `info` followed by the name of the command.

Info pages are more detailed than man pages. They are divided into different nodes or pages that we can read with an info reader, which works much like a web browser. To navigate with the info reader, use `'p'` for the previous screen and `'n'` for next screen within a page. To exit the info page, press `'q'`.

To learn more about man or info, you can type `"man man"`, `"man info"`, `"info man"`, or `"info info"` in the command line within your terminal.

CHAPTER SIX

Linux File System



OBJECTIVES

- Understand the difference between the Windows and Linux file systems
- Be able to navigate through the file system
- Be able to move, copy, and delete files and directories
- Understand hard and symbolic links
- Understand wildcards and case sensitivity
- Understand file archiving in Linux

From its inception, Linux was designed as a multiuser operating system. This means that any number of users can simultaneously work on a single machine. These users can connect to the system via different terminals or network connections while maintaining a separate storage area for personal information and individual desktop configurations.

Among the users working on a machine, Linux also distinguishes between different kinds of user roles. A user can log into a Linux machine as a normal user or as the root user, a superuser with privileges that authorize them to access all portions of the system and to execute any necessary administrative tasks.

All users, including the superuser, have their own home directories where all the user's private data (like documents,

bookmarks, and emails) are stored. While a user can modify anything in their own home directory, only the superuser can modify the system directories that hold all the centralized configuration files and executable files on the system.

To modify these files and folders, a user can choose to use a GUI-based file manager within the desktop environment or through a shell in the command line environment. The traditional method on Linux and Unix systems was to use the command line environment, but the GUI-based method is easier for most end users. While using the command line is often faster, it does require some deeper knowledge of the commands and their syntax to list, create, delete, or edit files and their properties.

Within a Linux system, all files and directories are in a tree-like structure. The topmost directory is the root of the file system, which is notated as a single forward slash (/). Note that the root of the file system is not to be confused with the root user or the root user's home directory (/home/root). In a Windows system, the root of the file system is designated as a backslash (\) and usually denoted by the drive letter C (usually written as C:\).

All the other directories in Linux can be accessed from the root directory and are arranged in a hierarchical structure using parent-child relationships. Unlike Windows, Linux does not use backslashes to separate the components of a pathname, but instead relies on forward slashes. Therefore, the private data of a user in Windows might be stored in C:\users\diontraining\documents, but in Linux it might instead be stored under /home/diontraining/documents. Additionally, Linux, unlike Windows, does not use drive letters. Instead, a user can detect which directory, drive, device, or network they are accessing from the appearance of the pathname itself. For example, you might access the storage directory on the secondary hard disk drive by using /dev/disk1/storage/.

Key File System Features

Another crucial difference to understand between Windows and Linux is the concept of mounting and unmounting partitions, drives, and directories. When Windows detects partitions and drives during the boot process, it assigns a drive letter to them. However, Linux partitions or devices are usually not visible in the directory tree unless they are mounted. Mounting a device means that it will become integrated into the file system at a specific location within

the directory tree. If a device isn't mounted, a normal user cannot access any data on the device or its partitions.

Luckily, most modern Linux distributions automatically mount partitions and devices for their end users. During the installation, users can define partitions to automatically mount when the system is started. Removable devices are usually also detected and mounted automatically by the system when they are connected. Desktop environments such as KDE or Gnome will notify the end user when a new device is mounted and becomes accessible. If you are using an older version of Linux and need to mount the device yourself, you will use the mount and umount commands. To learn about the exact usage of these commands, please use the man pages for them on your Linux system.

Unlike Windows, Linux distinguishes between uppercase and lowercase letters within the file system. Therefore, if you change the file test.txt to TeSt.txt or Test.txt, Linux considers all three different files. This case sensitivity holds true for both files and directories within the file system. So, if you have Letters and letters as directories, Linux would treat them as two different directories within.

Another major difference between Linux and Windows is the use of file extensions to define what a file is used for. For example, in Windows a file will end in .txt if it is a text file. In Linux, files may have a file extension, such as .txt, but it is not required. Instead, the operating system looks at the contents of the file to determine which application can read it instead of simply relying on the three-letter extension. Many users still add the three letter extensions, though, since it makes it easier for users to identify a file's type and use.

Like Windows, Linux does distinguish between normal files and hidden files. In general, the system uses hidden files to hold configuration details. Hidden files are indicated by a dot in front of the file name, such as .hiddenfile. In order to access a hidden file, you can switch the properties or options for the view in the file managers or use certain command syntax options in the shell.

As stated earlier, Linux is a multiuser system and, therefore, every file must belong to a specific user and a group. Only the owner of a file or directory (or the root user) can grant other users the permission to access them. Linux distinguishes between three different types of access permissions: write permissions, read

permissions, and execute permissions. Users can only access a file or a folder if they have at least a read permission to it. There are several ways to change the access permissions of files and folders: either via the shell or with the help of the desktop's file manager.

Although each Linux distro has its own way of doing certain things, all Linux developers recognize the need for some standardization in the layout of their directory structures. To address this need, the Filesystem Hierarchy Standard (FHS) was created. The FHS makes an important distinction between shareable and unshareable files. Shareable files may be reasonably shared between computers, such as user data files and program binary files. Unshareable files contain system-specific information, such as configuration files. The FHS makes a second important distinction between static files and variable files. Static files do not usually change except through direct intervention by the system administrator. For example, executable program files are considered static files. On the other hand, users, automated scripts, servers, or other devices can change variable files. For example, most of the files stored in a users' home directory are variable files since the user can easily change their contents.

Navigating Files and Directories

Now that we have an overview of the file system, it is time to learn how to navigate around it from the command line. But, before you can manipulate files and directories, it is important to understand which files are in which directories.

The **ls** command, short for list, provides this information. If no command line options are used with the command, the output will simply display the files in the current directory. However, additional options and file or directory specifications can be provided to display more detailed information. For example, using the **"-a"** option displays all files including hidden files, such as the files and folders which begin with a dot. If you instead use the **"-l"** option, it displays a detailed output, which includes permission strings, ownership, file sizes, and file creation dates in addition to the filenames.

Another useful option is the **"-d"** option, which is used when working with directories. When working in a directory that holds many subdirectories, using a wildcard with **ls** that matches one or more subdirectories may get an unexpected result: the output will

show the files in the matched subdirectories, rather than the information on the subdirectories themselves. To get information on the subdirectories rather than the contents of those subdirectories, you can instead include the “-d” option. If you want to learn about the `ls` command in more detail, type `man ls` at the prompt to display the man page for the `ls` command.

The `cd` command, short for change directory, changes the current directory to another one provided by the user. Although the current directory doesn't matter for many commands, it will matter when referring to certain file manipulation commands. When working with the `cd` command, do not forget that Linux uses a forward slash as a directory separator, as opposed to the backslash (`\`) used in Windows for this purpose. The forward slash and backslash are not interchangeable, so it is up to the user to use the proper one for the operating system they are using. In fact, the backslash has a different meaning in Linux: a backslash serves as a “quote” or “escape” character to enter otherwise hard-to-specify characters like spaces as part of a filename. Typing `cd` followed by a forward slash and the directory name or path to change to will make the shell switch to that directory. Do take note that depending on the distro and shell being used, changing the current directory may not actually reflect in the prompt displayed by the shell. To know the complete path of the current location, we can always enter the `pwd` command (present working directory) at the command prompt.

Remember that Linux uses a unified directory tree. This means that all the files on the system are located relative to a single root directory (`/`), which is represented by using the single forward slash character.

Files and directories can be referenced in three ways: absolute references, home directory references, or relative references. An absolute reference allows a file to be referenced relative to the root directory, such as in `/home/user1/file.txt`, which refers to the `file.txt` file in `user1`'s home directory. Absolute references always begin with the single forward slash since they start at the root of the file system.

A home directory reference uses the tilde character to refer to the user's home directory. If a filename begins with that character, it is as if the path to the user's home directory has been substituted for the tilde. For `user1`, `~/file.txt` is equivalent to the absolute

reference of `/home/user1/file.txt`.

A relative reference refers to a file reference that is made relative to the current directory. Every Linux directory includes two special hidden subdirectories known as dot (`.`) and dot dot (`..`).

Dot (`.`) refers to the current directory. This is also known as the present working directory. Personally, I always refer to it as “here”. Therefore, if you want to create a relative path, you can use the single dot as your starting point within a command or file’s paths.

The dot dot (`..`) refers to the parent directory. This is the directory one level above the current directory. Again, this is used heavily when creating relative paths to different files and directories across the filesystem. For example, if user2 is working in the directory `/home/user2/`, user2 can refer to user1’s home directory by using `../user1`. This references the directory above `/home/user2` (`/home`) and then the subdirectory user 1 within that directory.

Manipulating Files

We can manipulate files by using many different commands. Files can be created, copied them from one location or folder to another, moved, renamed, or even deleted from the file system using different commands within the shell.

Normally, files are created by programs or commands that manipulate them. For example, a graphics program like GIMP might create a new graphics file. The exact process to create a file differs depending on the program, but most GUI programs typically use a menu option called Save or Save As to create and save a file. Text-based programs within the command line often provide a similar functionality, but the details of how it is done vary greatly from one program to another.

The most basic file creation program is `touch`. The `touch` command is a command line program that creates a new, empty text file at the location provided as a parameter when the command first runs. To create a new file, simply type “`touch`” followed by the name of a file; for example, type “`touch newfile.txt`” to create an empty file called `newfile.txt` within your present working directory. Normally, it is not necessary to create a file of a particular type, since a program will automatically create the matching the file type needed for its file format. Sometimes, though, it is helpful to create an empty file just to create a few disposable files to test some other

command while you are learning your way around the shell and its usage.

Another command that is useful in working with files is the `cp` command. This command, short for copy, is used to copy files from one directory to another. It can be used in three different ways: to paste an option for a source filename and a destination filename, a destination directory name, or both. The `cp` command also provides many other options that modify its behavior. To learn all the details about the `cp` command, please type “`man cp`” in the shell to get a list of all the options and corresponding functions.

If you want to move or rename a file, we use the same command for both. This command is the `mv` command, which stands for move. The `mv` command uses syntax and usage that is like the `cp` command. If a filename is specified with the destination, the file will be renamed while being moved. If a filename is specified, and the destination directory is the same as the source directory, the file will be renamed but not moved. The `mv` command’s effects are much like `cp`, except that the new file replaces the original file instead of simply duplicating it.

Also, when the source and target files are on the same filesystem, `mv` rewrites directory entries without physically moving the file’s data on the disk. When a file moves from one filesystem to another, though, `mv` copies the file to the new disk and then deletes the original file from the source disk. To learn more about the `mv` command, type “`man mv`” from the command prompt to view its man page.

Sometimes, files are no longer needed on a system. When this occurs, we can delete these files the `rm` command, which stands for remove. By typing “`rm filename`,” the file in the current directory will be deleted from the filesystem. To delete an entire directory tree, use the `-r` option along with a directory name. This `-r` stands for recursive and causes the directory and all its files to be deleted. Another useful option, the `-i` switch, allows the `rm` command to prompt before deleting each individual file. This switch is a safety measure to make sure that the files are really intended to be deleted and is especially useful when using `rm` within an automated script. There are several other options for the `rm` command, and to learn more about them please type “`man rm`” at the command prompt.

One thing that is important to realize about the `rm` command is that it does not implement any kind of functionality like the recycle

bin within the Windows operating system. Once a file or directory is deleted with `rm`, it is gone and cannot be recovered except by using low-level forensic file recovery tools, which are well beyond the scope of this introductory course in Linux. Users should always be careful when using the `rm` command, especially when using the `-r` switch or when running the command using root privileges.

Hard and Symbolic Links

In Linux, sometimes it is handy to refer to a single file using multiple names rather than creating several copies of the file. To do this, a user can create multiple links to a file. This is similar to the concept of using a shortcut within Windows. In Linux, there are two types of links supported by the operating system: hard links and symbolic links. We can create both types by using the `ln` command, known as the link command.

A hard link is a duplicate directory entry where both entries point to the same file. Because the actual directory entry and the link both connect to the low-level filesystem data structures, hard links can exist within a single filesystem. When using a hard link, neither of the filenames holds any priority over the other. Instead, they are both tied directly to the file's data structures and data. To create a hard link, type "`ln filename linkname`," where `filename` is the original name of the file and `linkname` is the name of the link to create.

A symbolic link, also known as a soft link, is a file that refers to another file by name. This means that the symbolic link is a file that holds another file's name and, when you tell a program to read to or write from a symbolic link file, Linux redirects the access to the original file. Because symbolic links work by filename references, they can cross filesystem boundaries. To create a symbolic link, type "`ln -s filename linkname`," where `filename` is the original name of the file and `linkname` is the name of the new link to create. This symbolic link is similar to the idea of what most end users are familiar with, as it works like a shortcut within the Windows filesystem.

Wildcards

When working with files and directories within the command line, a user may want to refer to multiple items simultaneously. To do this, utilize a wildcard. A wildcard is a symbol or set of symbols

that stands in for other characters. Wildcards can refer to files and is also sometimes called globbing. Linux utilizes three different types of wildcards when referencing items in the filesystem: the question mark, the asterisk, and bracketed values.

To better explain the use of these three types of wildcards, pretend you have a directory containing five different files all starting with the letter B: ball, bell, bowl, bull, bale.

The question mark can replace a single character. For example, if I enter the command “ls b??l” then each file that starts with the letter b, has two other letters, and then ends with the letter l will be returned. In this case, the words ball, bell, bowl, and bull would all be returned since they meet the criteria, but the word bale would not.

An asterisk can replace any single character or multiple characters. For example, if I enter the command “ls b*” then any file that starts with a b will be returned since the * can replace all the letters after the b. In this case, the words ball, bell, bowl, bull, and bale would all be returned.

Bracketed values can replace any character from within the brackets into the search term. For example, if I enter the command “ls b[aou][lw]l” any files that match the letters from within the brackets would be returned. In this case, the words ball, bowl, bull would be returned since they can be made by replacing the second and third characters with characters from within the brackets.

Wildcards are implemented by the shell and are then passed to the command. For example, when the user enters the command “ls b??l”, the shell matches b??l with the three files bowl, ball, and bull outputs the results of the command as if the user entered “ls ball bell bowl bull”.

Please take note, however, that you need to be extremely careful when using wildcards. If you are careless, wildcards can lead to unexpected and sometimes undesired consequences. For example, copying two files using wildcards to another directory but forgetting to give the destination directory may result in copying the first file while overwriting the second and, in essence, deleting it.

Case Sensitivity

Linux’s native filesystem is case sensitive. This is a unique characteristic of Linux compared to other operating systems like

Windows, where case insensitivity is commonplace. Since Linux utilizes case sensitivity, this means that filenames are not treated as distinct and separate files whenever they contain the exact same characters but have different type case.

For example, a single directory can hold files called `afile.txt`, `Afile.txt`, and `AFILE.TXT` and, even if the content of each is exactly the same, the filesystem still considers each as three distinct and different files due to their different type cases.

Case sensitivity is primarily a function of the Linux filesystem and not of the operating system itself, though. If a user only uses media that is formatted within Linux, then their files will always follow this case sensitivity. But, if a user is accessing a non-Linux filesystem from some removable media, a non-Linux partition on a dual-boot computer, or using a network filesystem, it is highly likely that case-insensitive rules will apply.

For example, if you are accessing a FAT or NTFS formatted volume (which is commonly used in Windows), then case insensitivity will apply, and the Linux operating system can understand and use that case insensitivity without any issues.

But, to further complicate things, some shells (like `bash`) and programs always assume case sensitivity, even if they are accessing a case-insensitive filesystem. Due to this, it is a best practice to always treat your files with case sensitivity regardless of the operating system or filesystem format you are using if you are working with Linux and other operating systems.

Manipulating Directories

Most users are familiar with folders within a filesystem since most GUI file managers represent directories by using file folder icons. In the Linux filesystem, folders are technically called directories. Linux naturally provides both GUI-based and command line tools to manipulate directories located within the filesystem. These include directory-specific commands that allow users to create and delete directories, as well as to perform the file-manipulation commands discussed earlier, such as `list`, `copy`, `rename`, and `move`.

The `mkdir` command, short for make directory, is used to create a directory. To create a new directory named `newfolder` in the current directory, simply type “`mkdir newfolder`” in the command prompt. This is like right-clicking the mouse and selecting New

Folder within the GUI on a Windows machine. For more options with `mkdir` command, please refer to the `mkdir` man page.

The `rmdir` command, short for remove directory, destroys or deletes a directory. To delete a directory named `newfolder` in the current directory, simply type “`rmdir newfolder`” in the command prompt. When using `rmdir`, remember that file and directory names are case sensitive in the Linux filesystem, so it is important to specify the correct case to avoid accidentally deleting the wrong directory.

The `rmdir` command can only delete empty directories, though. If a directory contains any files or directories, the `rmdir` command will fail. The `-p` option, however, can delete a set of nested directories, if none of them hold any non-directory files.

To delete directories containing files anywhere within the directory tree, you can instead use the `rm` command with the recursive (`-r`) option. Because it can delete directories that are non-empty, `rm` with the `-r` option becomes potentially dangerous since it can delete files not intended to be deleted. Therefore, it is considered a best practice to first execute the `ls -r` command to display the directory and its contents before attempting to run `rm -r` to ensure that you are fully aware of the files and directories you are about to delete.

As stated before, the filesystem considers directories a special file type. These special files simply hold the user’s other files. Because of this, most of the file manipulation tools covered previously can manipulate directories, too. There are some caveats to this general rule, though.

First, the `touch` command cannot create a directory. Instead, the command `mkdir` makes a new directory. If the `touch` command is executed on a directory, it will refresh and update the directory’s time stamp instead.

The `cp` command can copy a directory, but only if the user specifies the `-r` or `-a` options to copy the directory and all its contents. Similarly, the `mv` command can move or rename a directory if the right syntax and options are utilized.

When creating links in the Linux filesystem, the user can only create symbolic links to a directory using the `ln -s` command. This is because the Linux filesystem does not support or allow hard links to a directory.

Archiving Files

A file-archiving tool collects a group of files into a single package file. This allows us to easily move the single file around on a single filesystem; back up the file to a recordable DVD, USB flash drive, or other removable media; or transfer the file across a network. Linux supports several archiving commands to perform this function, but the most prominent ones are tar and zip.

The tar program's name stands for tape archiver. Even though it was originally designed for backing up files to backup tapes, tar can back up or archive data to the filesystem, a hard drive, or removable media. The tar program is a popular tool for archiving various data files into a single file, called an archive file, while the original files remain safely stored on the disk. Because the resulting archive file can become quite large, it is often compressed via the tar program into what is known as a tarball. In fact, tarballs are often used for transferring multiple files between computers and are considered the universal method for distributing source code within Linux. The tar program is a complex command with numerous options, but the most common options are all that is needed for most users.

When running the tar command, the user must use only one command with at least one qualifier or option. To create a tar archive, enter the command `"tar -c tarfile.tar file1.txt file2.txt"`. This command will create a new tarfile called `tarfile.tar`, which contains the two text files: `file1.txt` and `file2.txt`. To create a tarball, a compressed tar file, enter the command `"tar -czf tarfile.tar.gz file1.txt file2.txt"`. This command will perform the same functions as the previous one but will also compress the file using gzip to reduce the file size. To learn about all the functions of the tar command, please consult its man page.

Three other common programs can compress individual files, rather than creating compressed directories like tar creates. These programs are gzip, bzip2, and xz. They take the original file, compress it, and create a new file with the same name but add a file extension to indicate the type of compression format.

Unfortunately, most programs cannot read a compressed file, so the file must first be uncompressed using the proper program prior to other programs using the file.

If a file was compressed by gunzip, it will have a `.gz` extension

in its filename. To uncompress a .gz file, simply use gunzip.

If a file was compressed with bunzip2, it will have a .bz2 extension in its filename. To uncompress a .bz2 file, simply use bzip2.

If a file was compressed with xz, it will have a .xz extension in its filename. To uncompress a .xz file, you must use unxz.

CHAPTER SEVEN

Searching and Extracting Data



OBJECTIVES

- Understand how to redirect output using pipes
- Be able to redirect operators using grep
- Be able to search for patterns within data files

The command line environment is an extremely powerful tool. As previously discussed, wildcards can substitute for a single character, multiple characters from a provided subset, or any character for part of a file name. This same concept can also search for data within a file or files, as well.

In this chapter, you will learn how to search for and extract data within various files and commands using redirection and wildcards.

But first, it is imperative that you understand how to link multiple commands together using the concepts of command redirect and input/output redirection.

Command Redirection

Command redirection allows a user to reroute the standard input to a second program or command. This is performed using a pipeline or command line pipe. A pipe is created by using a vertical bar (|) between the two commands. This key is usually found above

the Enter key on the keyboard and accessed by pressing the Shift key down while pressing the backslash key.

Pipelines can be useful when applied in various ways. For example, the lengthy output of the cat command (which displays the contents of a file to the screen) can be piped together with the less command to show only one page of content at a time. This avoids jumping to the last page of the file and not seeing the content in the rest of the file. The more command can also display one page of content at a time when combined with cat, but it operates a bit differently than the less command. To learn more about cat, less, and more commands, please consult their dedicated man pages.

By using pipes, a user can perform two or more commands in sequence and run them consecutively. This allows for quick execution of commands and complex tasks, resulting in less pauses in execution as well, since the system won't have to wait for the user to input the next command.

The grep command is commonly used in pipelines to search for keywords within the output of a given command. This command searches for files that contain a specified string and returns the name of the file and, if it is a text file, the line containing that string. The grep command can also search within a specified file for a specific string. To use grep, a user simply types the command's name, grep, an optional set of switches and parameters, a regular expression, and an optional filename if desired.

If you are planning to enter the world of system administration or cybersecurity, you will need to get comfortable using the grep command. This command is a security operation center analyst's best friend for finding information from across different log files contained in a Security Information and Event Management server. In-depth usage of the grep tool is beyond the scope of this book, but to learn more about the command please consult its detailed man pages for the complete list of options it supports.

Input/Output Redirection

In addition to redirecting the output of one command into another by using the pipe, a user can also redirect the input/output of a command to other locations. For example, if you need to save a program's output for future reference, you can redirect that output to a file. Similarly, the input to a command can be redirected from a

file, instead of requiring the user to enter it by using the keyboard.

Although input redirection may sound strange, some programs rely on this feature to enable them to process data. For instance, raw text files fed through a program search the text for specified patterns. In addition to redirecting output to files or input from files, a program's output can be passed to another program as an input through the process of piping, as covered earlier in this chapter. In addition to all of this, there is a related technique that involves the `xargs` command, which enables a user to generate command line options from files or another programs' output.

A Linux system has two type of output that are supported by default: standard output and standard error. The standard output is used for normal program messages. For example, when you run the `ls` command, the output is sent to the standard output and, therefore, displayed on your screen. The standard error, on the other hand, outputs error messages. By default, this also displays on the screen, but users can redirect these messages if desired.

There are several redirection operators available to more fully control where information is output to, displayed to, and/or saved to.

`>` creates a new file containing the standard output. If the specified file already exists, it will be overwritten. For example, if the command "`ls > contents.txt`" is executed, the output of the `ls` command will be saved to the `contents.txt` file and nothing would be displayed to the screen.

`>>` appends the standard output to an existing file. Just like the command above, we could redirect the output of the `ls` command to the file `contents.txt`, but instead of overwriting the file, we would simply add our output to the end of the current file.

`2>` creates a new file that will have the standard error redirected to the file. If the file already exists, its contents will be overwritten.

`2>>` appends the standard error to an existing file. If the file doesn't already exist, a new one is created. If it does exist, the standard error output will simply be added to the end of the file.

`&>` creates a new file that contains both the standard output and standard error output. If the file already exists, its contents will be overwritten.

There are also several redirection operators available to more fully control where information is input from in the shell.

`<` sends the contents of the specified file to the command in place of the standard input. For example, if I had a text file called `input.txt` that contained a single line of text that said `b??l` and entered the command `"ls < input.txt"`, the output would be equivalent to entering `ls b??l` at the command prompt.

`< <` enters a sort of interactive mode, where the following lines entered will be redirected to standard input. This will allow you to enter new information and redirect it to the same command, one line at a time.

`< >` specifies that a file will be used for both the standard input and standard output.

Now, while some of these, like the `< >` option, may seem like something you would never use, remember that Linux treats every device as a file that is mounted somewhere within its filesystem. So, an option like `< >` may be useful when reading or writing to an external hard disk, sending or receiving data to/from a sound card, or other such cases.

As a normal user, you likely won't use `< >` very often, but if you go on to become a Linux developer then it may become more useful to you at that time.

Let's put these concepts into practice using the `grep` command. For example, if a system administrator on the root account wanted to search for data about a particular user (Jason) within every configuration file in the `/etc` directory, they may do so quite simply by typing `"grep Jason /etc/*"`. Running the command without redirection, though, the output of the `grep` search would simply be displayed to the screen and not stored for future reference. Also, since our system administrator was executing the command as the root user, `grep` is likely to return a long series of outputs to the screen, making it long and difficult to read.

Therefore, the system administrator should instead employ redirection. To accomplish this, the system administrator should use a redirection operator and specify the filename where the data will be redirected into and stored. For example, entering `"grep Jason /etc/* > Jason.txt"` would send the output of the `grep` search to the `Jason.txt` file in the present working directory.

Another reason for using redirection is to change where the standard error is sent. For example, if the system administrator entered `"grep Jason /etc/*"` as a normal user instead of as a root user, the files that contain the username will appear on the screen,

but so will a lot of error messages. This is because a normal user lacks the permission to read many of the files within the /etc directory. By default, the standard errors are also routed to the standard output (the screen).

To remove all the errors so the system administrator won't have to see them cluttering up the screen, they can be redirected to a special device that exists on every Linux system, known as /dev/null. This null device is essentially an area where we can send garbage for the operating system to delete.

So, if the system administrator enters the command “grep Jason /etc/* 2> /dev/null”, then the grep command will search for the name Jason in every configuration file in the /etc folder, and redirect all of the errors messages to /dev/null so they are never displayed on the screen and are simply discarded.

Regular Expressions

While entering a search term like a username is useful, the real power of grep comes from combining it with regular expressions, also known as REGEX. Regular expressions are a way to describe patterns that a user might want to look for within a given data file or files. Many Linux programs allow for regular expressions and this provides the user with the tools for expressing patterns within text. Similar in principle to wildcards, regular expressions can specify multiple filenames or other text-based content.

There are two forms of regular expressions: basic and extended. Which regular expression form is assumed will depend upon the program that is being executed. Some programs accept just one expression form, while others can use both types.

The simplest type of regular expression is an alphabetic or alphanumeric string, such as HWaddr or Linux. These regular expressions match any string of the same size or longer that contains the regular expression. For instance, the HWaddr regular expression matches the following three phrases that might appear within a file being searched: “HWaddr”, “This is the HWaddr”, and “The HWaddr is unknown”. But the true power of regular expressions comes when a user inserts special characters that can denote patterns.

Bracket expressions are commonly used when conducting searches in grep using regular expressions. A bracket expression uses the square bracket characters ([]) to constitute a wildcard

search that selects any one of the characters within the bracket. For example, the regular expression of `b[aeiou]g` represents five words: bag, beg, big, bog, and bug.

A range expression also uses the square brackets, but, instead of listing every character within the brackets, it relies on using the starting and ending points of the range separated by a dash (-). Generally, using a range expression is easier than a normal bracket expression when searching for larger sets of data. Ranges can also be applied to alphabetic characters, like `[g-t]` which would represent all the letters g, h, i, j, k, l, m, n, o, p, q, r, s, and t. Remember that since Linux is case sensitive, the ranges `[a-z]` and `[A-Z]` are considered two different sets of ranges.

The single dot (.) is a wildcard that represents any single character except for the newline character, which is used when the return or enter key is pressed on a keyboard. For example, if the user entered `b.g`, that would represent `bAg`, `bag`, `bBg`, `bbG`, `b2g`, and so on, where any letter, number, or character could be inserted between the b and g since the single dot can represent any character.

Often when searching for data in a log file or configuration file, the user will need to search for where the line starts or ends. To denote the line's start, use the caret (^). To denote the end, use the dollar sign (\$). For example, if a system administrator wants to find any file that contains the phrase *and so it begins* at the end of a line, they would enter `"and so it begins$"` as the regular expression.

Another specification for regular expressions is to denote whether to make a full or partial regular expression match. If an asterisk (*) is used, it will denote zero or matches. If the asterisk is used with a dot (.*), then a substring match is instead made using the regular expression.

Finally, users must remember to use a backslash to escape any special characters in their regular expression search criteria. For example, the dot (.) is a special character that normally denotes a single wildcard character. But how would a user search for an IP address within the configuration files, since IP addresses must contain the dot character? For example, the user cannot simply enter `192.168.1.1` as the regular expression, because those dots would be replaced with any other single character by the operating system. To overcome this issue, we use a process known as escaping. Escaping a special character will bypass its normal special

function and instead treat it just as its regular text-based equivalent. For example, to search for the IP address of 192.168.1.1, you would simply escape each dot by placing a backslash in front of it. For example, to search for all the IP addresses starting with 192.168.1 and ending in 1, 2, or 3, the user can use the regular expression `192\.168\.1\.[1-3]` in their search.

CHAPTER EIGHT

Scripting Basics

```
#!/bin/bash
# This is a basic bash script.
a=Hello
b="Good Morning"
c=16

echo $a
echo $b
echo $c
```

OBJECTIVES

- Understand how to use a basic text editor
- Be able to create a basic shell script
- Be able to create scripts with conditions and loops

As you have already seen, the command line environment is an extremely powerful tool, but it becomes infinitely more powerful when you can automate numerous functions at once using a script. In this chapter, you will be introduced to the concepts of creating your own scripts using the bash scripting language. To create a script, though, you must first learn how to create a file using a text editor to enter all the script's commands and functions.

Text Editors

There are a lot of files in a computer system, but the most basic and flexible of them all is the simple text file. In fact, configuration files and shell scripts are typically just text files.

Linux users will often modify configuration files and create shell scripts. Therefore, it is necessary to understand exactly how to edit text files from within the shell. To edit text files, the user must be able to start the editor and either create a new text file or edit an existing file.

A text editor is a program that simply lets you view and edit documents stored in a plain-text format. Text files can include simple ASCII text; therefore, they cannot contain any graphics, fonts, emphasis on words (such as bold, italics, or underline), or any other advanced word processing features that you may be familiar with from a GUI-based program like Microsoft Word, LibreOffice, or even Google Docs.

Text files contain multiple lines of text that can vary in length from 0 characters to multiple pages. These text files can hold many different data types as well, but most commonly they contain human readable ASCII text.

If a user is viewing or editing their own files, they can do this as the standard user. If the user is attempting to view or edit system configuration files, though, then they must act as the root user by either using the `su` or `sudo` commands.

Text files generally contain five different types of data within the files, including human readable language (such as English, Spanish, French, etc.), programming language (bash, C, Python, etc.), formatted text, program or system configuration files, and log files.

There are a multitude of text editors that can be run within the Linux shell, but the four most popular are `vi`, `emacs`, `pico`, and `nano`.

One of the oldest and most popular text editors is `vi`, which is short for “visual”. The `vi` program is a text-based editor originally created for the Unix operating system in 1976. It is quite popular with Linux administrators who often edit configuration files, but it is a bit more difficult to use than some of the other more modern text editors.

`Emacs` is an extensible, customizable, self-documenting, and real-time display editor that can be used both from the shell and the GUI. `Emacs` is a much larger and more complex program than `vi`, so it is often not installed in lightweight Linux distributions by default. `Emacs`, like `vi`, was also first created for Unix back in 1976, and it still remains popular with Linux administrators.

`Pico` was designed to work similarly to `emacs` but removed many of the advanced features from within `emacs` to create a smaller program and file size. This made `pico` more popular with lightweight Linux distributions, but it's not nearly as full featured.

Some users found that `emacs` had too many features (and, therefore, was overly large), while `pico` didn't have enough features.

Nano was developed to fill in this gap in the market between emacs and pico. Nano has more features than pico, but it doesn't have as many as emacs. This also gives nano a smaller file size than emacs and allows it to be small and lightweight, while still maintaining the most needed features from emacs.

Each of the mentioned text editors has its own conventions for displaying information on the screen, manipulating text, and so on. Most text-mode text editors are similar up to a point; for instance, one or more lines at the top or bottom of the display typically show summary information or brief command prompts. Refer to the man pages of these text editors or launch them and explore their menus to know more about their behavior.

After launching a text editor and making changes to the file, do not forget to save the file or make a back-up copy. It is worth taking note that since these programs do not feature autosave functions, it is best to regularly save changes to prevent data loss due to interruptions.

Additionally, you should launch each of these programs, create a sample file, and get familiar with the usage of each of these programs to determine which text editor is your favorite—personally, I have always been a bigger fan of pico over vi.

Using Vi and Nano

Scan QR code to watch a video for this topic



There are many text-based editors within Linux, and these are helpful when we need to edit a configuration file or just quickly access a file to make some changes.

To run vi, type vi and the filename to be created. For instance, to create the file test1, type vi test1. This file starts blank unless it is an existing file. After typing the command and pressing enter, the left portion of the screen will be lined up with tildes (~). This signifies that those lines have nothing on them. This open file is also in edit mode. To start typing, press the "i" key on the keyboard to go into insert mode and start typing characters. Pressing enter moves the cursor to the next line. However, to go back to the previous line or any other character in the file, the arrow keys do not work; pressing the escape (ESC) key brings back edit mode, allowing for movement using the up and down arrows. Pressing the "x" key deletes characters and pressing the "i" key again allows us to insert or typing characters.

There are a couple of useful shortcuts to more easily manipulate

vi in edit mode. The caret (^) brings the cursor to the beginning of the line. Typing a number and w, for example "3w", brings the cursor three words forward while a number and b, for example "2b", brings the cursor two words back. To skip lines, enter a number and capital G; for example, "50G" brings the cursor to the 50th line in the file. Pressing the ":" key then typing "set nu" replaces the tildes with line numbers to more easily identify lines. To undo a previous action, typing "u" will undo it. To save the file, press ":" then type "w" to save the information to the file or "wq" to save and quit. Typing "q!" at that prompt discards any edits and quits vi. Vi's man pages will show more commands on how to properly use it.

Running nano is just the same as vi; typing nano and the filename brings up the editor and opens the file. Once nano shows up, it looks more like a command program. It shows more information than vi. For example, the program's name and the version is at the top left. The file's name is at the top center. The file's contents is also shown, if not blank. At the bottom are some commands. Commands that start with a caret (^), such as ^G, are entered by pressing the CTRL key and then hitting the G, and this brings up Help, which is like a man file. Compared to vi, nano is already in edit mode. The arrow keys work, and you can insert and delete characters just like in a word processor. And, similarly, text can be copied and pasted. It also has a spell checker and has undo and redo. Changes on the file are saved by pressing ^O, and ^X exits the program. With these features, nano has more functions and is more convenient than vi, and we can use it for those quick file edits and configuration changes.

Shell Scripting

A program written in an interpreted language, typically associated with a shell or a compiled program, is known as a script. In Linux, many scripts are created as shell scripts since they are associated with bash or another shell. Shell scripts are written by users to help automate tedious, repetitive tasks or to perform new and complex tasks. Most distributions use built-in scripts to perform many of the startup functions in a Linux system. Mastering scripting is important for administrators because it will help in managing the startup process of a system.

Shell scripts are plain-text files that are created in text editors such as vi, emacs, nano, or pico. A shell script begins with a line

that identifies the shell that should be used to run it. Consider the following simple script:

```
#!/bin/bash
echo "Hello World!"
```

The first line begins with two characters (`#!/`). These two characters are a special code that tells the Linux kernel that this is a script and that the rest of the line should be treated as a pathname to the program that will interpret this script. In this script, it is the path `/bin/bash`, which is the path to the bash shell program.

Within a shell script, the pound symbol or hash mark (`#`) is considered a comment character. This causes the script utility to ignore this line, but the kernel still reads it. On most Linux systems, `/bin/sh` is a symbolic link that points to `/bin/bash`, but it can also point to another shell. If a user specifies the script as using `/bin/sh` instead of `/bin/bash`, this almost certainly guarantees that any Linux system will be able to execute the script since they all have a shell program to run the script. If it is a simple bash script, the script can be run in most other shells. Unfortunately, if the script is more complex, then it will need a specific shell or else it could fail.

After writing the shell script, we must modify the text file to ensure it is now an executable file within the operating system by using the `chmod` command. The use of `chmod` will be covered in Chapter 12 (Ownership and Permissions).

As you begin to create shell scripts that are lengthier and more complex, you will want to have a good text editor program. If using a graphical desktop, KWrite and gnome's `gedit` editors are excellent for creating text shell scripts. However, if working from a command line, then `vi`, `emacs`, `pico`, or `nano` can be used instead.

Using Commands in Scripts

The simplest use of a script is to run a series of commands without user intervention. Commands built into the shell and external commands can both be used in scripts. This means that other programs can also be run from within a script.

Most of the commands that a user enters in the shell prompt are external commands. All these programs are located in `/bin`, `/usr/bin`, and other directories within the filesystem. These programs, as well as internal commands, can be run by simply using their names within the script. If needed, we can also specify parameters for each of these programs in a script. For instance, consider the following

script for copying a file from Downloads to Documents, launching an xterm window, and then starting up the KMail mail reader program inside the GUI:

```
#!/bin/bash
cp ~/Downloads/file.txt ~/Documents/file.txt
/usr/bin/xterm &
/usr/bin/kmail &
```

Aside from the first line identifying it as a script, the script looks just like the commands that a user would type into the shell to accomplish these three tasks manually. The only exception is that this script lists the complete paths to each program. This is usually not strictly necessary (as shown by the line with the `cp` command), but listing the complete path ensures that the script will find the programs even if the `PATH` environment variable changes. If the script produces an error when run (such as “No such file or directory error for a command”), you can help locate the program by running the “`which`” command to locate the path. For example, if you needed to find the path to the `xterm` program, entering “`which xterm`” at the shell prompt should return “`/usr/bin`” as its path.

In the basic script provided, notice that each line in the example script ends in an ampersand (&). This character instructs the shell to go on to the next line without waiting for the first command to finish executing. If the ampersand was not provided, the first `xterm` will open, but the `KMail` program will not open until the `xterm` program is closed.

Although launching several programs from one script can save time during the startup of a system or program, there are some situations where the user will want to wait for one line of the script to finish before starting the next line. This is common when running a series of programs that manipulate data from the previous command in some way. These scripts typically do not include the ampersands at the ends of the commands because one command must run after another or may even rely on the output from the first command.

While a script can use any command or program within it, there are certain commands that administrators commonly use within a script to conduct certain functions.

The `echo` command is probably the most used command in a script. The `echo` command displays a message to the user on the

screen within the terminal. For example, the script may start with a message to the user that says, “Please wait, the script is running...”. This type of message is displayed using the echo command.

When repetitive file maintenance tasks need to be automated, we use the ls (list), mv (move), cp (copy), and rm (remove) commands.

When text needs to be extracted from a specific field within a file, we heavily use the cut command. This is especially useful when working with a preformatted text file, like a command separated value (CSV) file.

The find command finds a particular file based on its filename, ownership, or other characteristics. If an administrator needs to locate a certain pattern of information within the contents of our files, they use the grep command instead. The grep command can also display the lines from within a provided file that match the search string.

Variables

Another way to expand the usefulness of your scripts is to use variables. A variable is a placeholder for a value to be determined at the script’s execution time. The value of a variable can be passed as parameters to a script, generated internally within a script, or extracted from the script’s environment. An environment is a set of variables, including items like the current directory or search path for running programs that a program can access.

Variables that are passed to the script are known as parameters or arguments. They are represented in the script by a dollar sign (\$) followed by a number. These numbers begin with \$0 (representing the name of the script), then go to \$1 (the first parameter), then to \$2 (the second parameter), and so on.

Consider the following script and try to determine what functions are being performed. If you do not understand a command, use the man command to learn more about it. When you think you understand the script, keep reading to see if you understood it properly.

```
#!/bin/bash
useradd -m $1
passwd $1
mkdir -p /shared/$1
chown $1.users /shared/$1
chmod 755 /shared/$1
```

```
ln -s /shared/$1 /home/$1/shared
chown $1.users /home/$1/shared
```

The script first creates an account using the first argument given to the script as the username. The script then changes the account's password and creates a directory in the /shared directory corresponding to the account name, which again is received as the first parameter from the user running the script. The ownership of this newly created directory is then given to this new user account, and its permissions modified to give the user read, write, and execute permissions. Next, the script sets a symbolic link to that directory from the new user's home directory and provides the new user account ownership over this folder in the home directory.

If you didn't understand everything in this script right now, that is perfect normal since some of these commands and programs are new to you at this point. By the end of this book, though, you should feel comfortable reading a script like this, or even writing one yourself.

To use this script, a user would simply type the name of the script, the new username, and enter the desired password twice, as shown in this excerpt below:

```
DionTraining:Documents jason$ ./user_script.sh jasondion
Changing password for jasondion
New password:
Retype password:
Password for jasondion changed by jason
DionTraining:Documents jason$
```

When the script runs, the only thing displayed to the screen is whatever the commands normally would display, such as the prompt for the new password to be entered. All the other commands are run in the background with no user interaction or knowledge. This is referred to as silent operation. For this reason, many script writers will add display notes into their scripts using the echo command so that information can pass back to the user during the operation. In effect, this script simply replaced seven individual commands with a single script. For each command, the first parameter (jasondion) was used as a variable to prevent the user from needing to reenter the new username each time.

Conditional Expressions

By default, a script will execute all the commands listed from the first line to the last line, but sometimes it is helpful to skip lines

based on certain circumstances. To achieve this, we use conditional expressions.

Conditional expressions enable a script to perform one of several actions depending on what conditions have been met or the value of a variable. The `if` command allows the script to evaluate a given conditional expression that is placed within brackets after the `if` keyword, and it then perform one of two different actions, depending on whether a certain condition is true or false.

For example, the file exists condition (`-f`) is true if file exists and is a regular file. The file exists and isn't empty condition (`-s`) is true if the file exists and has a size greater than 0. The "`string1 == string2`" condition is true if the two strings have the exact same values.

These conditional statements may also be combined with the logical AND (`&&`) or the logical OR (`||`) operators. When conditionals combine with the logical AND, then both sides of the operator must be true for the condition as a whole to be considered true. When we use the logical OR, if either side of the operator is true, then the condition as a whole is also considered true.

Consider the following script fragment that demonstrates the `if` command and conditional expression:

```
if [-f /tmp/tempfile.txt]
then
echo "/tmp/tempfile.txt was found; aborting script."
exit
fi
```

This basic conditional evaluates whether the file (`tempfile.txt`) exists within the `/tmp` directory. If the file exists, then the condition is true, and the "then" branch will execute to display the message using the `echo` command, and the script will exit. If the conditional instead found that the file does not exist, then the `echo` and `exit` commands are skipped, and the `if` condition ends when the `fi` command is reached. The `fi` command simply shows the end of an `if` command statement block, as shown above.

A conditional, such as the one above, may be useful if your script creates a file when it runs. If you check if the file already exists at the beginning of the script and it tests true, then this indicates that the script was already or is still running.

The `if` command also supports using the keyword `test` instead of the brackets, if desired. The following variant would also work:

```
if test -f /tmp/tempfile.txt
```

```
then
  echo "/tmp/tempfile.txt was found; aborting script."
exit
fi
```

Another useful test condition is to use the output of a command as the test case, such as the following pseudocode:

```
if [command]
then
  [additional commands]
fi
```

In the pseudocode above, the additional commands will only run if the command in the test condition executes successfully. If the command returns any error codes, then the additional commands simply would be skipped during the script's execution.

The if command also supports the ability to define what commands should run if the condition is not met. This is often read as "if, then, else". For example, you may hear programmers and script writers say, "If the condition is true, **then** do this, **else** (otherwise) do that."

```
if [command]
then
  [do this command]
else
  [do that command]
fi
```

This type of coding will cause only one of the two branches to execute: either the then or the else. This is useful when a programmer needs to allow the program to make a choice in which commands to run or what files to operate on based on a certain condition.

It is important to note that in the pseudocode above, multiple commands or lines can be added into each branch of the then-else, as needed.

While the if-then-else works well for conditions and choices with two outcomes, this is sometimes limiting. For example, if you create a script that displays four choices to the screen as part of a menu option to the user running the script, the user then enters a number, 1, 2, 3, or 4, and the script runs a certain command based on that number. Using the if-then-else command would require a fairly messy decision tree, as shown in this pseudocode:

```
if [$userchoice == 1]
```

```
then
    [do command #1]
else
    if [$userchoice == 2]
    then
        [do command #2]
    else
        if [$userchoice == 3]
        then
            [do command #3]
        else
            if [$userchoice == 4]
            then
                [do command #4]
            else
                echo "Error – User didn't enter 1, 2, 3, or 4."
            fi
        fi
    fi
fi
```

While this pseudocode is functional, these nested if-then-else statements are rather clumsy and confusing to understand. Luckily, shell scripting provides us with a suitable alternative, known as a case statement, for scenarios where there are multiple possible outcomes. Consider the following pseudocode using a case statement that performs the same function as the nested if-then-else above:

```
case $userchoice in
    1) [do command #1];;
    2) [do command #2];;
    3) [do command #3];;
    4) [do command #4];;
esac
```

Generally, a case statement uses a variable, and each pattern shown before the right parenthesis is a possible value of that variable. In the example above, we expect the user to input a value of 1, 2, 3, or 4. Each set of commands ends with a double semicolon to signify the end of the actions for that specific case. This allows multiple commands to be run from within each matching case to the value entered by the user in \$userchoice. At the end of the case, we use the keyword `esac`, which is case spelled backwards, just like `fi` was used for `if`.

When using a case statement, the variable (\$userchoice in the example above) simply finds the matching case (for example, 3) and executes that single line until it reaches the double semicolon. Once the double semicolon is reached, the script jumps to the line after the `esac` keyword. If no matching case value is found, then none of the statements will execute, and the script resumes operation after

the `esac` keyword.

Loops

So far, all the scripting components covered will only execute a single time. For example, if an if-then-else or case is considered and the test case doesn't match it currently, it will be skipped and never executed. Unfortunately, sometimes it is important to do things multiple times or to test a condition multiple times to see if the condition has changed. To do this, we use a special structure known as a loop.

A loop allows a script to perform the same task repeatedly until a particular condition is met or until a certain condition is no longer true. Consider the following loop and try to determine what it will cause when the user executes this script.

```
#!/bin/bash
for x in `ls *.txt`; do
    cat $x
done
```

In this example, the loop will execute once for every item returned by the command `ls *.txt`. Therefore, let's assume there are 3 files in the current directory: `foo1.txt`, `foo2.txt`, and `foo3.txt`.

When this script is executed, the `for` loop will first run `ls *.txt` and receive a list of 3 files that contain the file extension of `.txt`. Since there are three files, this loop will execute 3 times. The first time through the loop, the command `cat foo1.txt` is run. This will cause the contents of the `foo1.txt` file to display on the screen. The second time through the loop, the command `cat foo2.txt` will run, and the contents of `foo2.txt` will display on the screen. Then, the loop will run a third time, and this time the contents of `foo3.txt` will display to the screen. Since there are no more `.txt` files in the current directory, the loop will end.

Instead of using a command like `ls` to set the number of times the loop will run, the programmer can also use the `seq` command. The `seq` command generates a list of numbers, starting from the first argument and continuing up to the last argument. If three arguments are given to `seq`, then it will treat them as the starting value, the increment value, and the ending value. Considering the following output:

```
DionTraining:Documents jason$ cat loop.sh
#!/bin/bash
```

```
for x in `seq 1 3 10`; do
    echo $x
done
DionTraining:Documents jason$ ./loop.sh
1
4
7
10
DionTraining:Documents jason$
```

This output shows the contents of the loop.sh script . This is a bash script, as shown by the first line (`#!/bin/bash`). The loop uses the `seq` command to create the number of times to execute the loop. Each time the loop executes, the number (`x`) is displayed to the screen and then a new line is created. Since the `seq` command has three inputs, the counting will begin at 1, add 3 each time through the loop, and end at 10. The example above shows the output, with 1, 4, 7, and 10 displayed to the user's screen.

In addition to the `for` loop, there is also another type of loop called a `while` loop. A `while` loop executes the commands within the loop repeatedly as long as the condition being tested remains true.

```
#!/bin/bash
while [test condition]
do
    [commands to execute]
Done
```

There is another loop called an `until` loop. The `until` loop works like the `while` loop, except the `until` loop executes continually as long as the condition remains false.

Functions

Scripts also support the creation of functions within them. These functions perform a specific subtask and can be called by name from within other portions of the script.

Functions are created by placing parentheses after the function name, then enclosing the commands that will run as part of the function within curly brackets, as shown in the pseudocode below:

```
#!/bin/bash
myfunction() {
    [commands to execute]
}
```

In addition to the name (`myfunction`) shown above, the script's writer can optionally add the keyword `function` (as in, `function myfunction`) to identify that section of the script as a function, but

this is completely optional. Once the function is created in the script, it can be referenced from anywhere else in the script as if it was an ordinary internal or external command.

Functions are useful for creating modular scripts and can increase the readability and efficiency of a script. For example, if the script needs to perform the same computation ten different times throughout it, the computation can be placed in a function, which can then simply be called directly when needed.

Please note, a function cannot run outside of the script since it isn't a true program or command. Also, functions do not run in the order they appear in the script, but instead they run anytime their name appears within the script's main body.

The Exit Value

Every script, like every good story, has a beginning, a middle (or body), and an end. That end, though, can come before the script reaches the very end of the script's content, as demonstrated in our if-then-else example earlier in this chapter. Regardless of when the script ends, it will return an exit value back to the shell to indicate it finished its execution.

A script's default return exit value is the exit value of the last command the script executed. This is represented in the shell by the variable `?`. The exit value can be controlled, though, and it is possible to exit from the script at any point by using the exit command. Used without any options, exit causes immediate termination of the script, with the default exit value of `?`. This can be useful in error handling or in aborting an ongoing operation when needed. If the script detects an error or if the user selects an option to terminate, the script can include an exit call to quit prior to the script crashing.

If the script instead sends a numeric value between 0 and 255 to the exit command, then the script terminates and returns the specified value as the script's exit value. This is a useful feature that can signal a particular error if the script is being called by another script or program. For example, the script may exit with a value of 1 to signify that an invalid input was provided to the script, or a value of 2 to signify that an output file doesn't exist. The options are limitless and simply left up to the script's writer to decide the values and their meaning.

For example, the script's designer might create a variable

(\$termcause) to hold the value returned during the script's termination. When the program first executes, the value of \$termcause will be set to equal 0, and then, if the script detects a problem that will cause termination, it will reset \$termcause to a non-zero value. Any numeric codes may be used here since there is no set meaning for such codes by default. Whenever the script reaches a condition where it should exist, it will simply use "exist \$termcause" to exit and return the value of the \$termcause variable back to the shell as the exit value.



It is time to put together previously discussed script components into a complete basic script. To create a simple script, we will use the nano command and create a script named logic.sh by typing "nano logic.sh" at the terminal prompt. Start by commenting the first few lines using hashtags or the pound (#) sign. In this case, `#!/bin/bash` and Simple logic test display.

```
#!/bin/bash
# Simple logic test and display
```

Start by using an if command. If the first argument is going to equal 1 (\$1 = '1'), the system will display the next statement (then) by using echo "The argument entered was 1". An exit is placed, and the fi command ends the loop. The second if command in the script shows that if the next argument is going to equal 2 (\$1 = '2'), the system will display the next statement (then) by using echo "the argument entered was 2", and an exit is placed. Next comes an else command to be used for cases other than the previous two arguments (the number entered is neither 1 nor 2) to display by using echo "the argument entered was not 2". It is also exited, and an fi ends the loop.

```
if [ $1 = '1' ]
then
    echo The argument entered was 1
    exit
fi
if [ $1 = '2' ]
then
    echo The argument entered was 2
    exit
else
    echo The argument entered was not 2
    exit
fi
```

Make sure to save the script by pressing ^O, and then press ^X to exit to the terminal prompt. Change the permissions of the file to make sure it can run. Type "chmod 755 logic.sh" to make the file executable. Typing "./logic.sh 1" should satisfy the first if statement and should display "The argument entered was 1" and so on. However, this simple script assumes that there is an argument entered and that there is only one argument. Otherwise it will throw errors when no argument is made, such as typing "./logic.sh". It will return an error:

```
diontraining@work: ~/Documents/scripts$ chmod 755 logic.sh
diontraining@work: ~/Documents/scripts$ ./logic.sh 1
The argument entered was 1
diontraining@work: ~/Documents/scripts$ ./logic.sh 2
The argument entered was 2
diontraining@work: ~/Documents/scripts$ ./logic.sh 3
The argument entered was not 2
diontraining@work: ~/Documents/scripts$ ./logic.sh 0
The argument entered was not 2
diontraining@work: ~/Documents/scripts$ ./logic.sh
./logic.sh: line 4: [: =: unary operator expected
./logic.sh: line 10: [: =: unary operator expected
The argument entered was not 2
diontraining@work: ~/Documents/scripts$
```

The first thing to do is to create a check for the number of arguments. If it doesn't equal 1 ($S\# \neq 1$), then it will display the usage instructions by using echo. This also exits, and the loop ends with the fi command. This portion of the script checks how many arguments there are. If you don't have exactly one argument, the system quits the script. Now the script should run without throwing any errors, no matter how many or what argument is supplied.

```
#!/bin/bash
# Simple logic test and display
if [ $# != 1 ]
then echo "Usage - This script requires one argument that is a number."
    exit
if [ $1 = '1' ]
then
    echo The argument entered was 1
    exit
fi
if [ $1 = '2' ]
then
    echo The argument entered was 2
    exit
else
    echo The argument entered was not 2
    exit
fi
```

There is a bunch of things that we can do with scripts, as scripts can be as simple as the one mentioned here, or they can carry out complex logic flows. By adding in these logic steps when creating a script, we can test conditions by using if, then, and else to find out what is going on and to let the computer do the work.

CHAPTER NINE

Packages and Processes



OBJECTIVES

- Understand how package management systems function
- Be able to install, update, or remove a program
- Be able to identify running processes
- Understand memory usage, logging, and kernel execution

Linux itself is just the operating system, but most users focus on the programs they use to get their work done. When a user has a new type of work to do, they often will install a new program to handle that function.

Most users do not compile the program directly from source code themselves, but instead install and upgrade their software using a special program called a package management system or a package manager.

Once a program is installed and executed, it starts up multiple smaller functions known as processes. Each of these processes perform different functions, such as accessing the internet, writing files to a disk, and various other tasks. Each of these processes take up some memory on the system, and the operating system carefully monitors and logs each action.

An Overview of Package Management

Package management is an area of Linux that varies between distros, but there are a few principles in common amongst all flavors. First, every package is created as a single file. Linux package files, unlike Windows installers, are not programs. Each of these individual packages rely on other programs to do the work of installing the software into a Linux system. These installation programs are commonly referred to as package management systems or package managers.

Each package also contains dependency information. This information indicates to the packaging software what other packages or individual files must be installed for the package to properly install and to ensure that the resulting program will operate correctly. To help with this installation process, each package also contains version information so that the package manager can determine which version of each package is the most up to date.

Unlike some operating systems, we can install Linux on multiple different types of architectures that have different processor instruction sets, which converts software into processes for the hardware to execute. Due to this, each package must contain the architecture information to identify which CPU type the package is intended to be installed on. This is normally either i386 (for Intel processors) or AMD (for AMD processors), but other variants like RISC and ARM can also be found in some cases. Each piece of software being installed is compiled into a binary package (or executable package) using one of those specific processor instruction sets, and, therefore, they cannot be installed on a system with a different processor.

The package management system maintains a database of information about all the installed packages on a given system. This information includes the names and version numbers of all the installed packages, as well as the locations of all the files installed from each package. This information enables the package software to uninstall software quickly, establish whether a new package's dependencies have been met, and determine whether a package that a user is trying to install has already been installed and, if so, whether the installed version is older than the one being installed.

Package Management Systems

There are two most common package management systems in use in Linux today are RPM and Debian. These systems differ in various technical details, in the commands used to manage their packages, and in the format of the package files they use. An RPM package cannot be installed on a Debian-based system, and vice versa. In fact, installing a package intended for one distro on another is a bit risky even when they use the same package type because a non-native package may have dependencies that conflict with the needs of native packages within a given system.

When package managers were first created, all package management systems worked locally on a given system. To install a package, a user would have to first download a package file from the package creator's site or from some other internet-based resource. After it's downloaded, we could install the package with a local command. This approach is tedious when a package has numerous dependencies.

If we attempted an installation and then found unmet dependencies, the user would have to search out and download several more packages. Then, after finding that one or more of those packages also has unmet dependencies, the cycle goes on and on. By the time all these dependency packages have been tracked down and installed, a user may have had to install a few dozen additional packages just to install the one program they wanted to use.

Thankfully for us, most modern distributions provide network-enabled tools to help automate this laborious process. These package management tools rely on what is known as software repositories. These repositories provide a central collection point from which the package managers can automatically download packages and their dependencies. In practice, managing software in Linux now simply requires using text-mode or GUI tools to interface with a centralized software repository for any distribution.

In modern Linux systems, there are five basic steps to installing a new piece of software. First, we issue a command to install a program. Second, the package management software locates all the program's dependencies and notified the user of any additional software that they must install. Third, the user issues an approval for the package manager to download and install all the dependencies on their behalf. Fourth, the software downloads all the necessary dependencies and installs them. Finally, the package

manager installs the program and returns control of the system back to the user.

Upgrading software works in a similar way, although upgrades are less likely to require downloading a lot of dependencies. Many distributions will automatically check with their repositories from time to time and notify system users when updates are available. By doing so, the system is kept up to date by simply clicking a few buttons when the user is prompted to do so.

Removing software, unlike installing or upgrading, can be done entirely locally by the package management software. This is because the package management software continually keeps track of all installed packages and their dependencies, so they can be removed when requested by the user.

The user should run the package management software as a root user or by using the `sudo` command. This is because the software needs root access to fully install all the dependencies and modify system configuration files, as needed. When updating the system, the package management will automatically prompt the user to enter the root password when the system software requires it.

Using RPM and DEB Packages

Scan QR code to watch a video for this topic



As previously mentioned, RPM and Debian package management systems are two of the most commonly utilized package management systems in some Linux distributions. Ubuntu is a Debian-based distribution, so it uses the Debian-based package manager called apt or Advanced Packaging Tool. CentOS, Fedora, and other distributions under the RedHat family use the RPM package manager. In CentOS, this is called YUM (Yellowdog Updater, Modified) while, in Fedora, it is called DNF or Dandified YUM: a fork and improvement of YUM.

To work with the package manager DNF, first we must update the package list. Typing "dnf check-update" on a Fedora terminal tells DNF to check the repository and get a list of the latest version of all the programs that can be installed. It will then produce the list of the latest available versions. Then, "sudo dnf upgrade" checks if there are any programs in the system that need updates. The command is run as root since there might be system-installed programs that can only be updated with root privileges. As DNF

finds new versions of the programs to install, it will ask the user if the programs will indeed be updated. To search for a program to be installed in the system, typing `dnf search keyword`, and replacing 'keyword' with the keyword for the program, searches the repository for programs that contain that word. For example, typing `dnf search virtualbox` will search the package list for programs that have the word `virtualbox` in them. All the packages with the searched keyword will be listed on screen. The files listed on the screen come in `.rpm` file package format. To look into the details of a package, type `dnf info filename`, replacing `filename` with the package's name. To install a package, type `sudo dnf install package`, replacing `package` with the package name. We use `sudo` since there may be dependencies and additional software that need to be installed needing root privileges. To remove software, type `sudo dnf erase package-name`, replacing 'package-name' with the package name. In this uninstallation, we again use `sudo` since there may be dependencies and software to be removed that needs root access.

Ubuntu, being Debian-based, uses the `.deb` file package format. To update the package list, type `sudo apt update`. If there are new versions of the software installed in Ubuntu, typing `sudo apt upgrade` will update programs to the latest version available in the repositories. Sometimes, when doing an upgrade, `apt` will advise on programs that may be too old, just sitting in the system and not being used as a dependency by any other program, or aren't used at all. To save space and keep the system secure, these can be uninstalled by typing `sudo apt autoremove`. To search for a package in `apt`, type `apt search [package keyword]` replacing the words in the bracket with a keyword relating to the program to be installed. To view the details of a package, type `apt show [file name]`. Information that will be shown include the package name, version, priority, section, origin, maintainer, URL to the bug reports and resolution, etc. To install a package, type `sudo apt install package-name`, replacing 'package-name' with the package name. To uninstall that package or any other package, type `sudo apt remove package-name`.

The good thing about package managers like DNF and `apt` is that they automatically install other software, called dependencies, that are needed by the package currently being installed. They keep track of packages in the distribution's repository, as well as

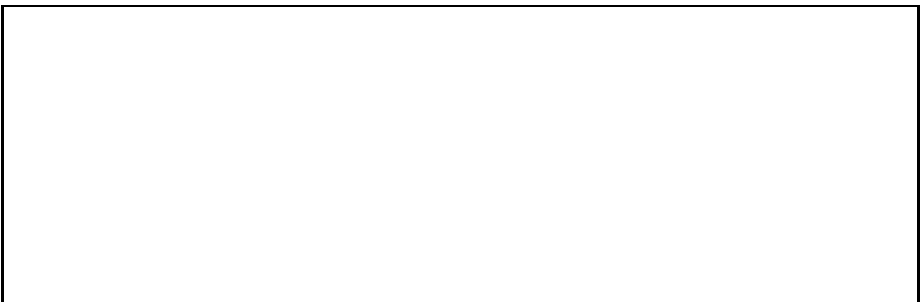
packages currently installed in the system, allowing for easy updates to newer versions and uninstallation. These package managers can also be easily accessed from the terminal especially for systems, such as servers, that may not have a GUI.

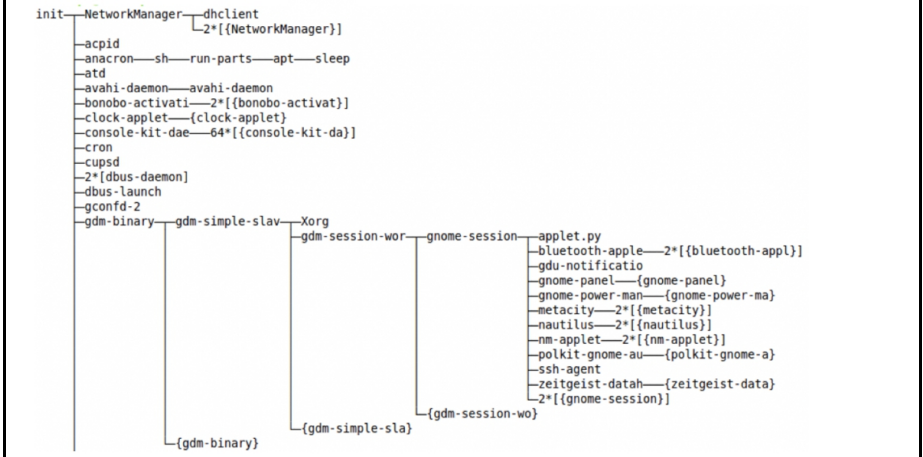
Process Hierarchy

Previously, we discussed that the Linux kernel is at the core of a Linux installation. The kernel manages memory, provides software with a way to access the hard disk, assigns CPU time to portions of a piece of software, and performs other critical low-level tasks. The boot process loads the kernel early on, and it is the kernel that is responsible for managing every other piece of software code that is running on a Linux computer. One of the many ways that the kernel imposes order on the potentially chaotic set of running software is to create a hierarchy.

When it boots, the kernel runs just one program, normally `/sbin/init`. The `init` process is then responsible for starting all the other basic programs that Linux must run, such as the programs that manage logins and always-on servers and services. These programs, if launched directly by `init`, are known as its children. `Init`'s children can in turn launch their own children. For example, when you log into a Linux system, a child process is launched to support that function. Therefore, anytime a process is launched, it is considered a child. The process which launched a given process is then called its parent. This parent-child relationship creates a natural hierarchy for all the running processes on a given system.

This results in a tree-like hierarchy of processes that, when illustrated, is often depicted like an upside-down tree. As shown in the figure below, there are a small subset of the many processes that run on a typical Linux installation. This image only displays a few processes associated with a text-mode login, including the login tool that manages logins, a couple of daemons, and a few user programs. A working Linux system will likely have dozens or hundreds of running processes, as shown here:





Identifying Running Processes

Every process has a process ID or PID number associated with it. These numbers begin with 1, so the PID for the init process is normally 1. Each process also has a parent process ID or PPID number, which points to its parent. Many of the tools for managing processes rely on these numbers, and particularly on the PID number to identify the running process.

Before managing processes, it is important to identify them. To do so, we use the `ps` and `top` command line utilities. With both tools, we can search for processes in various ways, such as by their name or by the resource that they utilize. One of the most common reasons for identifying a process is to know how much memory it is consuming, which can be done with the `free` command.

The simplest tool for identifying processes is `ps`, which produces a process listing. Given the large number of `ps` options, most users have their own favorite way of using the program. For example, typing `ps ax` is my favorite since it usually produces the information I need, such as the PID values, command names, and the command-line options for all processes on the computer. If you slightly alter the command by adding a `u`, as in `ps aux`, this will add usernames, CPU loads, and a few other details to the output produced by the `ps` command. The sheer scope of the information produced, however, can be overwhelming.

One way to narrow this scope is to pipe the outputted results through the `grep` command, which eliminates lines that don't include specified search criterion. For example, if you wanted to

only determine the PID number for the gedit process, you might run the command like this:

```
$ ps ax | grep gedit
27946 pts/8 S1 0:00 gedit
27950 pts/8 S+ 0:00 grep—colour = auto gedit
```

This example reveals that gedit has a PID value of 27946. This is usually the most important information when using ps, since the PID value can be used to change a process's priority or terminate it. If you need to terminate a process, you can do so using the kill command.

Although ps can return process priority and CPU use information, the program's output is usually sorted by the PID number. Also, it is important to remember that the ps command only provides information at only a single moment in time, much like a snapshot. So, if you run the ps command right now and then run it again in 3 minutes, you will get different results. To locate CPU or memory intensive processes quickly, or if there is a need to study how resource use varies over time, the top utility is a more appropriate tool. This utility is essentially an interactive version of the ps utility.

By default, top sorts its entries by CPU use, and it updates the display every few seconds. In order to determine if an application is behaving properly on a system, it is important to become familiar with the purposes and normal habits of the programs running on the system. This allows you to create a mental baseline of what normal function looks like on a given system. Because each program has different legitimate needs in terms of processing and memory usage, it is impossible to give a simple rule for judging when a process is consuming too many resources, but if you have a good baseline from which to begin your comparison, this can become much easier.

One of the pieces of information provided by top is the load average, which is a measure of the demand for CPU time by applications. The load average can be useful for detecting runaway or malicious processes. For instance, if a system normally has a load average of 0.5 but suddenly gets stuck at a load average of 2.5, it is possible that a few CPU-hogging processes may have hung or become unresponsive. Hung processes sometimes needlessly consume a lot of CPU time. Therefore, you should use top to locate these processes and, if necessary, stop (or kill) them.

Measuring Memory Usage

Processes consume important system resources like processor time and system memory. Using the `top` utility allows the user to sort the processes by CPU time by default to quickly identify the processes that are consuming the most processing resources. When `top` is running, there are several single-letter commands can be entered, some of which prompt for additional information. For example, if the user presses the `M` key within `top` sorts processes by memory use. To learn more about the `top` utility and all these single-letter commands, please consult the `man` page for the `top` utility.

If the `top` utility is currently being sorted by memory usage, it is quite easy to identify the processes that are consuming the most memory. As with CPU time, we cannot determine that a process is consuming too much memory simply because it is at the top of the list as some programs legitimately consume a great deal of memory.

For example, a simple text editor like `gedit` should consumed much less memory than a photo editing program like `gimp` since `gedit` manipulates much smaller files. But, if the same program usually requires only a few megabytes of memory, and today it is using a gigabyte of memory, this is something that we should further investigate investigated as abnormal activity and a possible indication of a runaway process.

Sometimes a program consumes too much memory, either because of inefficient coding or because of a memory leak. A memory leak is a type of software bug in which the program requests memory from the kernel and then fails to return it when it is done using the memory. A program with a memory leak consumes increasing amounts of memory, sometimes to the point where it interferes with other programs. As a short-term solution, the user can terminate the program and launch it again, which will reset the program's memory consumption. The problem will likely reoccur, but if the memory leak is small enough, then at least useful work can still be done.

To study the computer's overall memory use, the `free` command is a useful utility. This program generates a report on the computer's total memory status. The two most important lines within the resulting display of the command are the `Mem:` and the `Swap:` lines.

The Mem: line reveals the total RAM statistics. This includes the computer's total memory minus whatever is used by the motherboard and kernel, the amount of memory used, and the amount of free memory. Most of the computer's memory being used is a normal state since Linux puts otherwise unused memory to use as buffers and caches to help increase the speed for accessing information from the hard disk. Therefore, the Mem: line isn't the most useful by itself, but instead needs to be considered along with the -/+ buffers/cache: line that shows the total memory used by all the computer's programs.

The Swap: line reveals how much swap space Linux is using. Swap space is disk space that is set aside as a substitute for physical memory. Whenever Linux runs out of RAM or when it determines that RAM is better used for buffers or caches than to hold currently inactive programs, it will replace the use of physical RAM with a swap space contained as a file on the hard disk. Swap space use is generally quite low, and if it rises too much then performance problems can occur. In the long run, increasing the computer's physical memory

(RAM) is generally the best solution for a Linux system that is continually running out of memory, but a larger swap space can become a suitable temporary workaround, if needed. When suffering from performance problems because of excessive swap use, terminating some memory-hogging programs can help speed up your system. Also, memory leaks can lead to such problems and terminating the leaking program can restore system performance to normal.

Log Files

Many programs only run in the background and are not visible to the end user. These programs generally to run services for the network, such as providing DHCP, running a web server, or running an email server. These background services are known as daemons.

Since these programs run in the background, they continually require system resources, and the user can't actually see the details of what is happening in the background since their operations are not displayed to the screen. To determine what these programs are doing, a user must consult a log file, since these daemons commonly write information about their normal operations to text-based log files. Therefore, being able to find and read these log files is an

important part of diagnosing problems with a background service or daemons.

The first step in diagnosing a problem with a daemon is to locate its log file. In Linux, most log files are stored in the `/var/log` directory. For the LPI Linux Essentials certification exam, it is important to remember where log files are commonly store. While the following log files and directories within the `/var/log` directory are not an exhaustive list, they do makeup some of the most important log files on any Linux system: `boot.log`, `messages` or `syslog`, `secure`, `cups`, `gdm`, `secure`, and `Xorg.0.log`.

The `boot.log` file is used to summarize the services or daemons that start late in the boot process via SysV startup scripts. If your Linux system is having issues during the boot up process, you should review the `boot.log` file to see if an error message was recorded in the log file.

There are also general-purpose log files that contains messages from many different daemons on a given Linux system. These are known as `messages` or `syslog`. There log files store messages from various daemons when they do not have their own dedicated log files.

If your system is having an issue regarding security, such a log of which users attempted to use `su`, `sudo`, and other root privilege mechanisms, then you should check the log file known as `secure`. This file is located at `/var/log/secure` and is used for all security-related messages within the system.

The `/var/log/cups` directory is used to hold log files related to the Linux printing system. CUPS is an acronym for the Common UNIX Printing System, and it is a modular printing system for Unix and Linux systems which allows the computer to act as a print server. If you are experiencing an issue with printing from your Linux machine, you should review the logs within the `cups` directory.

The `/var/log/gdm` directory holds log files related to the GNOME Display Manager (GDM), which handles GUI logins on many systems. GNOME is a commonly used desktop GUI on many Linux distributions, so if you have an issue with the graphical user interface, then reviewing the logs within the `gdm` directory is a good place to start your troubleshooting efforts.

There is another graphical component on most Linux systems called the X Windows System, also simply known as X. The log file

for X is `Xorg.0.log`, and it contains all the information on the most recent startup of X on the system. If you are having a generalized graphics issue, then checking `/var/log/Xorg.0.log` is a good place to start your troubleshooting.

Because log files are constantly recording information about the system, its operations, and its errors, these files can grow very large in size. In fact, if the log files are not limited in size, they could completely fill up your hard drive and crash the operating system. To prevent this, information within the log files is frequently rotated. This means that the oldest entries within the log files are deleted and overwritten with newer entries.

Also, some programs will instead create new log files. they then rename the latest log file with a date or number, and the old log file is deleted once it reaches a certain age. For example, if the messages log file was rotated on July 1, 2019, `/var/log/messages` will become `/var/log/messages-20190701`, `/var/log/messages-1.gz`, or something similar, and a new `/var/log/messages` will be created. This practice keeps log files from growing too large.

Most log files are simply plain-text files, so they can be checked using any tool that can examine text files. While there are specialized programs that can parse, read, and compare logs, for basic log reviews a simple display program like entering the `cat` command or using a text editor like `gedit` will work just fine.

Some programs create their own log files, too. However, most programs rely on a utility known generically as the system log daemon `rsyslogd`. This daemon's process name is generally `syslog` or `syslogd`. Like other daemons, it starts during the boot process by the system startup scripts. Several system log daemon packages are available for you to use on a Linux system. Some of them provide a separate tool, like `klog` or `klogd`, to handle logging messages from the kernel separately from ordinary programs.

The behavior of the log daemon can be modified, including adjusting the files to which it logs certain types of messages, by adjusting its configuration file. The name of this file depends on the specific daemon in use, but it is typically `/etc/rsyslog.conf` or something similar, depending on your distribution.

Once running, a log daemon accepts messages from other processes by using a technique known as system messaging. It then sorts through the messages and directs them to a suitable log file depending on the message's source and a priority code.

Combining `grep` with log files can be a truly powerful combination for a system administrator. By properly using `grep`, you can quickly search through hundreds of thousands of lines within a given log file to find the exact issue or error you are trying to find.

Kernel Ring Buffer

The kernel ring buffer can be thought of as a log file for the kernel. However, unlike other log files, it is stored in memory rather than in a disk file. Like regular log files, though, its contents continue to change as the computer runs. To examine the kernel ring buffer, type `dmesg`. This can sometimes create an overwhelming amount of information, so the output is typically piped through the `less` utility.

```
$ dmesg | less
```

Alternatively, if the necessary information is associated with a string, you can use `grep` to search for it. For example, to find kernel ring buffer messages about the first hard disk, `/dev/sda`, type the following:

```
$ dmesg | grep sda
```

The kernel ring buffer's messages can be particularly difficult to understand, but they can also be invaluable in diagnosing hardware and driver problems since it is the kernel's job to interface with hardware.

The kernel ring buffer should be searched and reviewed if a hardware device is behaving strangely. Even if it is difficult to understand, a message from this log can be entered in a search engine or passed on to a more knowledgeable colleague for expert advice and further troubleshooting.

Some distributions place a copy of the kernel ring buffer when the system first boots in `/var/log/dmesg` or a similar file. This file can be reviewed if the computer has been running for a long time and its earliest entries are already lost from the copy within the memory's buffer. If you want to create such a within your distribution and it isn't already created by default, then edit the `/etc/rc.d/rc.local` file and add the following line to the end of the script:

```
$ dmesg > /var/log/dmesg
```

CHAPTER TEN

Networking Basics



OBJECTIVES

- Understand the basic network features in Linux
- Be able to configure a network connection in Linux
- Perform network testing in Linux
- Understand how to best protect your network in Linux

Networking is a critical part of modern computing. That's why most Linux distributions are capable of automatically creating a network connection when a user first boots up a Linux system. Unfortunately, there are some rare occasions when this automated process does not work properly. When this occurs, the user may need to uniquely configure the connection or debug a given problem.

Basic Network Features

Before you can understand how to configure, test, or protect a network connection, it is important to understand some of the most basic network features and terms. This includes commonly used network components like the internet, ethernet, Wi-Fi, TCP/IP, DHCP, an IP address, the network mask, the router, and DNS.

Most users are already familiar with the internet since they use it every day. In fact, when you bought this book online, you were

using the internet. The internet simply refers to the globe-spanning network of interconnected computers that use the standard Internet protocol suite (TCP/IP) to communicate.

To connect to the internet, most users will use a wired or wireless network. A wired network is often referred to by the hardware that network uses called ethernet. Technically, ethernet is a specific way of breaking the data down into frames of information and then transmitting those frames over a media (such as a wire or radiofrequency wave), but most people refer to wired networks as “ethernet networks”. This ethernet standard is defined within the IEEE 802.3 documentation standards.

If you are using a wireless network, though, this is defined in the IEEE 802.11 standard and referred to as Wi-Fi. While there are other wireless networking standards that exist, such as cellular and Wi-Max, the most common for desktops, laptops, and other Linux-enabled devices is Wi-Fi.

Regardless of whether your device uses a wired or wireless connection, it is most certainly using TCP/IP to transmit the data from your system to another. The Transmission Control Protocol/Internet Protocol (TCP/IP) is a set of standards that underlie most modern network communications at the software level. Every network connected device these days supports the TCP/IP protocol to communicate. For over 30 years, this has been the de facto standard, and it can be used with both wired and wireless connections.

In order for your device to communicate on a network using TCP/IP, that device must have a way for other devices to uniquely identify it. This can be done using its hostname or an IP address. A hostname is an alphanumeric name that a computer uses in order to be easily identified by humans. These hostnames consist of a computer portion and a network portion. For example, if you refer to the hostname of www.diontraining.com, you are really referring to the computer serving as our company’s web server (known as www) on the network (known as diontraining.com).

Unfortunately, though, computers don’t understand these alphanumeric hostnames since computers are logical, mathematical creatures. Therefore, each computer must have a unique Internet Protocol (IP) address that can be used to identify itself on a given network. An IP address is used by computer to communicate with each other easily.

There are two types of IP address formats, known as IPv4 and IPv6. IPv4 is the older format and utilizes a 32-bit address in the form of four decimal numbers separated by dots and each of those numbers should be between 0 and 255, such as 10.4.62.178. The issue with IPv4 addresses is that there are only 4.2 billion unique addresses. While that sounds like a lot (and it used to be), in modern networks like the internet have quickly depleted this amount of IP addresses.

To replace IPv4, a newer variant known as IPv6 was introduced. IPv6 uses a 128-bit address that consists of hexadecimal digits separated by semi-colons. Due to the much large bit size of the address, there are over 3.4×10^{38} IPv6 addresses available for use.

For the LPI Linux Essentials exam, it is not important to remember the number of IPv4 and IPv6 addresses, but you should be able to identify these addresses based on sight. For example, if you see 192.168.1.1, you should identify it as an IPv4 address, whereas if you see an address like fe80::afdc:48ff:fe00:5577, you should identify it as a IPv6 address.

So, if computers prefer numbers, but people prefer easier to read and remember hostnames, how can we associate both the hostname and the IP address together? To perform this function, a system known as the Domain Name System (DNS) was created. DNS is a global network of servers that automatically translates between hostnames and IP addresses for our systems. To use DNS, your Linux system simply needs to have the IP address of the DNS system added as part of the network configuration. Then, whenever the user types a hostname or domain name into an application, such as entering www.diontraining.com in a Google Chrome web browser, the application will ask DNS to convert the name (www.diontraining.com) into its assigned IP address, and then the browser can connect to the remote server using its IP address.

To properly configure your Linux system to use the network, it needs four pieces of information: its own IP address, its network mask, the IP address of its gateway, and the IP address of a valid DNS server.

In IPv4, when an IP address is assigned, it must also be provided with a network mask that identifies what network the host is contained within. By using a network mask (netmask), the system determines which portion of the IP address represents the host and

which portion represents the network. For example, if a user's system is assigned the IP address of 192.168.1.56 with a netmask of 255.255.255.0, this signifies that this host is (56) and is a part of the (192.168.1.0) network. The concept of subnetting heavily uses these conventions, but that is beyond the scope of this book and the LPI Linux Essentials exam. If you want to learn more about subnetting and netmasks, I highly recommend considering the CompTIA Network+ certification exam preparation courses or books.

The final component of the network configuration is the IP address of the system's gateway. Every system can communicate within its own network by default, such as the 192.168.1.0 network in the netmask example above. But, if the user wanted to communicate with another system in a different network (such as 172.15.12.65 or www.diontraining.com), then the traffic must be routed through a gateway.

A gateway is a device that connects two or more networks together. When a user connects to the internet, they are actually connecting through a gateway, most commonly their network's router. This router links to another network, which in turn links to another, and another, until it reaches the final destination. To contact a computer on another network, your computer communicates through its default gateway, which is assigned by configuring its gateway's IP address within its network configuration. Most home and small businesses networks simply use small broadband modem/routers combination device to connect to the Internet, and these devices perform routing, DNS, DHCP, and other useful services in one small, form factor unit.

Therefore, for your Linux system to actually be able to go online and connect to the internet, it needs to have four pieces of information configured within its network settings: its IP address, its netmask, the IP address of its gateway, and the IP address of its DNS server. This information can be manually or automatically configured on your system. Configuring this network information manually is time consuming and can often be complicated, but thankfully there is an automated method to do this which relies on a protocol known as the Dynamic Host Configuration Protocol (DHCP).

DHCP is an automated mechanism that most computers use to obtain their network configuration information from a server on

their network. This is extremely useful, especially in large scale corporate networks with thousands of computers that need to be configured.

By relying on DHCP, a system sends out a broadcast message on the network to discover if a DHCP server is available. When a DHCP server receives this discovery message, it automatically sends the requesting system an offer that contains an available IP address, a netmask, the gateway IP, and even the IP of a DNS server for the requester to use. Then, the requester will send back a request message to the DHCP server asking to use the network configuration information it just received from the server. This is done because the requester may have received multiple offers from different DHCP servers on the same network, so it has to request which one it wants to utilize. Finally, the DHCP server will send an acknowledgement message back to the requester once it accepts their offer. At this point, the requester now has all the information needed to automatically configure its own network connection and the DHCP process is complete. This process is known as DORA, or Discover, Offer, Request, and Acknowledge. For the LPI Linux Essentials exam, though you are not required to understand this four step process in-depth, you should remember that DHCP automatically configures a network connection within a Linux system.

Configuring a Network Connection

Most often, the network connection will be configured automatically, and DHCP assigns the vital network information. Sometimes, though, those details will have to be configured manually or the network interface must be enabled for it to function.

As stated earlier, a computer requires (at a minimum) two important pieces of information to connect to a local network: an IP address and a netmask. If you want the computer to also be able to perform name resolution and gain access to an external network, then DNS and the gateway must also be configured. Configuring every computer on a large network with all this information is time-consuming. Additionally, this manual configuration can lead to problems caused by human errors, such as mistakes created when entering in the IP address and other critical information. For this reason, most enterprise networks provide a DHCP server that

assigns all this critical information to the other computers within the network when they first connect.

Depending on the configuration of the server, DHCP can deliver IP addresses using either fixed or dynamic configurations. A fixed configuration ensures that the DHCP server provides each computer with the same IP address every time it boots up and connects to the network. A dynamic configuration instead has the DHCP server provide each computer with an IP addresses each time it boots up, but this IP doesn't have to remain the same each time.

Each computer can use either a wireless or wired network connection. Wireless connections are common on laptops, tablets, smartphones, and internet of things devices since they often lack a physical network cable due to their size or configuration. Some desktops, though, have Wi-Fi capabilities in addition to their wired connections.

To easily configure the Wi-Fi connection, it is best to use the graphical user interface. Each distribution uses a different method and setup for connecting to Wi-Fi, so you should consult your distribution's online manual or forums for details of how to do this. For most distributions, connecting to the Wi-Fi through the graphical interface works similarly to a Windows or Mac computer and relies on a point and click interface.

If someone needs to fine-tune the wireless connection, they can use various tools to scan for available wireless networks and manage the existing Wi-Fi links connected to a given system. To do this, the user can utilize the `iwlist` and `iwconfig` utilities. The `iwlist` command can identify nearby Wi-Fi networks by scanning the airwaves within the Wi-Fi frequency bands of 2.4 Ghz and 5.0 Ghz, depending on the type of wireless adapter installed within the user's system. To scan for a list of nearby networks, simply type `iwlist scan` or `iwlist scanning` as either the root user or using the `sudo` command to obtain a complete list of the networks within range of the system.

Once a network is found, a user can manually connect or disconnect from a specified network using the `iwconfig` utility. There are numerous options and flags that can be used with the `iwconfig` command, so I recommend consulting the man pages for this utility to ensure you are using it properly.

To configure a wired connection, a user may simply use the graphical user interface like they would within Windows or Mac. If

the user requires more fidelity and control over the configuration, then we may use command line tools instead. Just as `iwconfig` configures a wireless network, the command `ifconfig` configures a wired network. The `ifconfig` utility allows a network connection to be started up or shut down, allows for the allocation of an IP address and network mask to a specific piece of hardware, and allows the defining of other configuration details within a connection. The `ifconfig` command is primarily for wired connections, but it can configure a wireless connection once the wireless connection has been established using `iwconfig`.

There are many other flexible text-based tools for both wired and wireless connections. These tools include `route`, `/etc/resolve.conf`, DHCP utilities, and distribution-specific network scripts.

The **`route`** utility allows the user to adjust the computer's routing table to determine the path through which the network device will send specific network packets. By using the **`route`** command, a Linux system can effectively become a network router for the other machines on the network as well.

The `/etc/resolv.conf` file contains the IP addresses of up to three DNS servers as well as the name of the computer's Internet domain and of other domains that should be searched when the user omits a domain name from a hostname. If you need to modify the DNS entries used by a given Linux system, this can be done within the `/etc/resolv.conf` file.

As stated earlier, DHCP is the easiest method for configuring the network connection on a Linux machine since it automatically configures the system. To enable this, though, the Linux machine must have a DHCP client installed. The most common DHCP client programs are known as `dhclient` or `dhcpcd`.

The `ifconfig`, `route`, and DHCP client programs only produce a temporary change to the computer's network configuration. To make these changes permanent, the settings must be stored within a configuration file that is loaded when the system is booted up. To help automate this process, many distributions create network scripts to help with this boot up and configuration process. Each distribution uses its own methods and scripts for this purpose, and these are referred to as distribution-specific network scripts.

Testing the Network

In most cases, the network connection will work fine from the moment the Linux system first boots up. However, there are times when a user may have a need to diagnose a problem with the network because a formerly working connection has stopped working or it does not work from the start. There are several ways to test for network connectivity.

The first network connectivity test checks the routing table using the `route` command to ensure that the default route is properly set and to check that the routing is sensible. In most cases, typing `route` by itself at the terminal will display the routing table to the display. The routing table of a typical workstation or server is generally quite simple and can be automatically configured. If the Linux system is functioning as a network router, though, these often require complex routing tables which are beyond the scope of the LPI Linux Essentials certification exam.

The next most basic network connectivity test uses the `ping` utility. The `ping` command sends a simple network packet to the system you name, via IP address or hostname, and waits for a reply. In Linux systems, `ping` continually sends packets once every second until the command is interrupted with the `Ctrl + C` keystroke combination.

To diagnose where a network problem resides when using the `ping` command, you must consider three scenarios. First, if you can ping a system on the internal network but not a system on an external network, the problem is probably in your router or in an improper router specification. Stated another way, if you can ping another workstation within your network but you cannot ping a website like Google or Facebook, then the issue must exist with the router or the external network connection.

Second, if you can ping a system by its IP address but not by its name, then the problem is most likely with the DNS server or your system's DNS configuration settings. So, if you can successfully run `ping 8.8.8.8`, but you get an error when running `ping diontraining.com` then the issue is likely caused by a DNS issue.

Third, if you cannot ping any systems at all, then you probably have a fundamental configuration problem on your system. This could be caused by a corrupted network software stack or even faulty hardware on your Linux system. In any case, the problem exists within your machine, and not with the larger network or remote systems.

While ping is very useful in testing the entire connection from your system to the remote system, there is a more useful tool that can be used when you are trying to determine where exactly the connection is broken between your system and the remote system. To determine where the break in connectivity is occurring, a command known as traceroute is utilized. The traceroute command sends a series of three test packets to each system between your computer and a specified remote system. As traceroute conducts each set of tests, it displays the results on the screen to inform the user of which systems are reported as functional and which may be the cause of the break in connectivity. This is helpful for determining whether a problem in network connectivity exists within a network which you are responsible for or if it is problem with an external connection and, therefore, a problem for your internet service provider to solve.

As stated earlier, the problem may be caused by an issue with the domain name system (DNS). DNS is the internet's equivalent of a phone book; it maintains a directory of domain names and translates them to Internet Protocol (IP) addresses. This is necessary because, although domain names are easy for people to remember, computers or machines need to access websites based on IP addresses. DNS problems can cause networks to fail almost as badly as a physically cut cable. Because both people and many network tools rely on hostnames, if DNS resolution doesn't work then the network becomes nearly useless. A user can test their network's DNS server by using several tools, such as host, dig, and nslookup.

The final diagnostic tool to consider is known as netstat. This utility is sometime referred to as the Swiss Army knife of network troubleshooting tools because it can be used in place of several other tools depending on the parameters passed to it. For example, the --all or -a option displays information about the ports that a server has open and is currently listening for network connections on, as well as any already-open connections. To learn all about netstat, please enter man netstat from within your terminal window.

Protecting the Network

These days, one of the most important issues involving technology revolves around system security and data protection. In

this section, you will find a few basic tips to prevent your Linux system from becoming compromised by a malicious attacker. While many people claim that Linux is more secure than Windows, without proper precautions a Linux system can be just as vulnerable as its Windows counterpart. Luckily for us, Linux doesn't have nearly as many viruses or worms designed to run on its system, but by following these five simple rules you can increase your chances of keeping your system safe from attackers.

First, you should always shut down any unused services or daemons on your Linux system. In Linux, the majority of risks center on an outsider's ability to break into a system by abusing a server program that is running on your machine. Therefore, it's important not to run any servers, services, or daemons unnecessarily. Many distributions automatically install and run numerous servers, such as the Secure Shell (SSH), the Apache web server, a Sendmail server, or a Postfix email server. To thoroughly remove a server from a system, you should uninstall it by using the appropriate package management system by using the `uninstall` command.

Second, you should always enable a firewall on your Linux system. A firewall is a software program or system setting that inspects each network transactions and then allows or disallow them based on programmed criteria within an access control list (ACL). Most Linux distributions enable an embedded software firewalls by default, but there may be a need to adjust their settings for your specific needs. When configuring a firewall, it is best to use a "deny by default" security posture, and only allow network connectivity in and out of the system as needed for your operational use.

Third, you should always use good passwords for your accounts on your system. If your system runs a login server of any type, then setting a long, strong, and complex password can minimize the risk of an outsider breaking in by simply guessing one of your user's passwords. Passwords should be at least 14 characters long, include uppercase and lowercase letter, numbers, and special symbols. Also, your everyday user account and your root user account should never use the same password.

Fourth, always be ssuspicious when using a network connected system. You must always remain suspicious of any untrusted sources of data, such as emails, websites, and other internet sources.

Phishing is a common occurrence; this is an attempt to extract sensitive data from users by posing as a trusted individual or organization via email or other means. Phishing and similar attacks can trick users into providing their passwords, financial data, and other critical data. While malware (malicious software) is rare on Linux system, a determined attacker could create malware to target your organization. Therefore, it is best to stick to official software sources and always remember that emails, websites, and other types of communication can be easily faked to trick you.

Fifth and finally, always keep your software up to date. Whenever a hacker finds a way to exploit a software bug for their advantage, the creators of the software fight back by releasing a security update to the software program. But, if you don't update the software on your system, then you are still using the older software with the bug and are vulnerable to attack. Therefore, it is critical to always keep your software patched and up to date using the package management tools described previously in this book. It is best to check regularly for software updates and most distributions will have an automated mechanism to check for updates daily or weekly.

Connecting to a Network

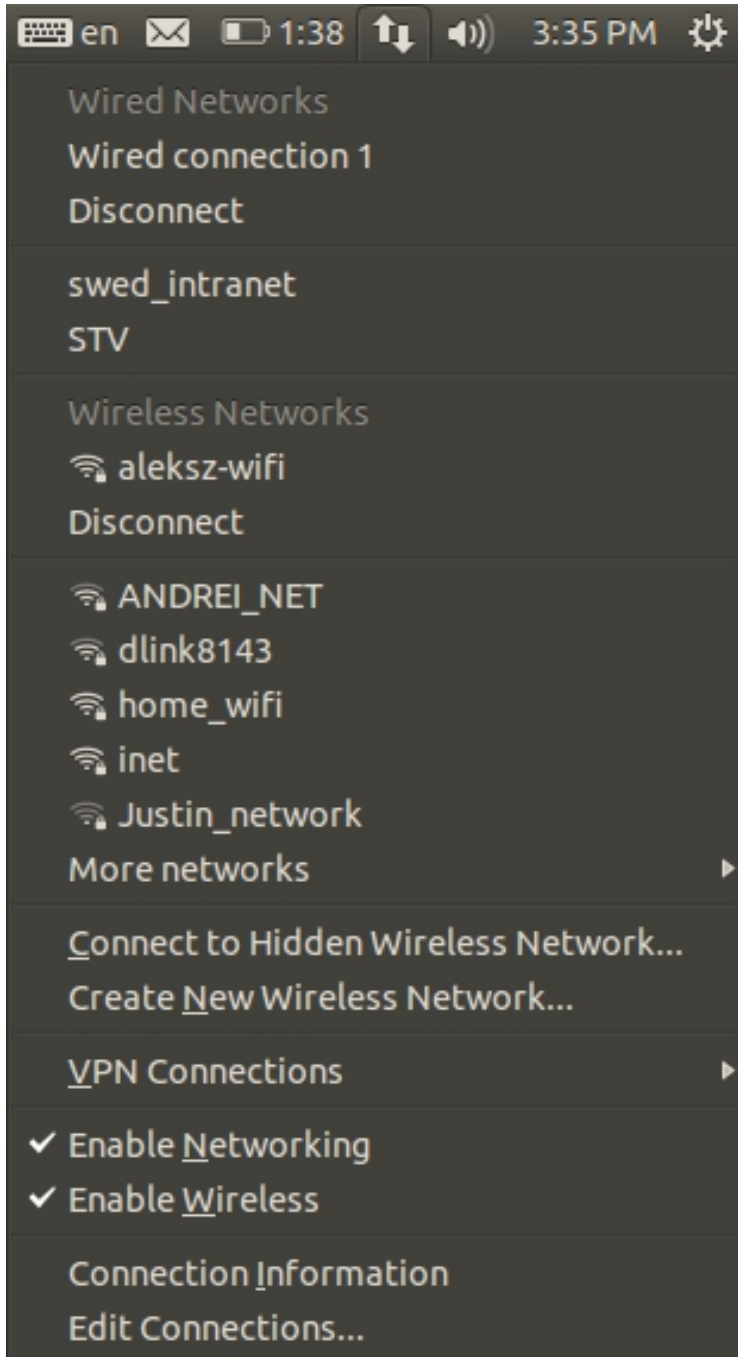
Scan QR code to watch a video for this topic



There are two ways to connect to a network in Linux: by using the GUI or by using the command line. The easiest way is through the GUI. In Ubuntu Unity, the network icon, showing wired and wireless connections if available, is at the top right corner of the taskbar. Everything from there will be fairly straightforward. Clicking on the network icon will bring up menus that show options to configure both wired and wireless connectivity. This figure shows that this machine is connected to both a wired (Wired connection 1) and wireless (aleksz-wifi) network.

There are options to disable the wired connection (by clicking on Enable Networking) or disable the wireless connection (by clicking on Enable Wireless). Connecting to a wireless connection may need the passphrase of the selected SSID.

To use the command line, open a terminal and issue ip commands. Typing "ip address show" shows the current ip address of the system. There will be several adapters that will show up



address -

The next adapters may show the wired or wireless connection,

whichever is present. Each adapter section will show information such as the name of the adapter, its MAC address, the ip address assigned to it, etc. When troubleshooting a local network issue, ip addresses wouldn't be of much concern. All you would need to see are the links in layer 2, the MAC addresses. To do that, type "ip link show". The ip link specifically focuses on layer 2, switching, as opposed to ip address, which shows layer 3, routing. To turn off an adapter (just like when disabling Networking from the network icon on the desktop), type "sudo ip link set adapter-name down". Replace 'adapter-name' with the adapter name to be switched off from the "ip address show" command. Typing "ip address show" again after turning the adapter off will show its state as DOWN and that there are no layer 3 routing information or ip addresses assigned to it. To turn it back on, simply type "sudo ip link adapter up". To manually assign an ip address to a network adapter, type "sudo ip addr add ip-address dev adapter-name". Replace 'ip-address' with the ip address and subnet mask to be assigned to the adapter and replace 'adapter-name' with the name of the adapter. Conversely, to remove that assigned ip, type "sudo ip addr delete ip-address dev adapter-name". More information about the ip command can be found in its man pages.

CHAPTER ELEVEN

User Accounts and Groups



OBJECTIVES

- Understand how user accounts work in Linux
- Describe basic account security
- Understand group management in Linux
- Describe the difference between the root user and other users

Linux is a multiuser operating system, which means that a single Linux computer can support many users, each with their own unique account. In this section of the book, we will discuss important account principles and the commands for working with these accounts. We will also cover the root user account, with its special privileges for performing most system administration tasks. Additionally, we will discuss groups, which are logical collections of user accounts that can be given special permissions within a Linux system.

Understanding User Accounts

By utilizing user accounts, multiple users can share a single physical computer while maintaining a logical separation for their programs and data files. Since each person is assigned a unique user account, system administrators can track who is using system

resources and who is taking which actions within a Linux system.

Account features for each user's account are defined in the `/etc/passwd` file. This file contains colon-delimited lines where each line contains a single record that defines an individual user's account. The information contained in the fields of these records includes the username, password, UID, GID, comment field, home directory, and the default shell for that user.

The first field contained within the colon-delimited line is the account's username. This is the most relevant feature, since it uniquely identifies the user to which all the other attributes will be applied against. Linux account usernames consist of alphanumeric characters, but by convention most usernames are simply created using lowercase letters. Underscores, periods, and dashes are also valid characters in some Linux distributions. If including the dollar sign special character, it must appear at the end of the username.

The next field is the password. Each user account is typically protected by a password, which is required to log into the computer. When a user enters their username and password, this creates a successful authentication and approves access. Originally in Linux, the password was stored in this field, but due to security issues the password field is now replaced with a single `x` instead. The `x` character indicates to the system that the actual password for the user will be stored in the `/etc/shadow` instead of the `/etc/passwd` file, as the `/etc/shadow` file is more secure since only the root user can access that file. When storing the password in the `/etc/shadow` file, it is first hashed using a one-way algorithm and then salted to ensure that no one can read the user's password from the `/etc/shadow` file.

The username provided for each account is just a human readable label that the computer uses to identify each user account on the system. In reality, the computer tracks these users using a numeric identification number known as the user identification (UID) number. UID numbers always start at 0 and continue upward based on the number of accounts created on the system. The root user is always UID 0. In most distributions, the user accounts will begin with UID number 1000. So, if a user created two accounts on a brand-new Linux installation as `jason` and `diontraining`, these accounts would be UID 1000 and 10001, respectively. UID numbers below 1000 are reserved for system and service account.

After the UID field, the group identification (GID) number is

displayed. This GID is associated with one or more groups, and each group can contain one or more users. Just as directories and folders hold numerous files, a group collects numerous users who all have the same function or require the same permissions in a system. One of the best practices in system administration is to always assign permissions to a resource by assigning it to a group and not to an individual user. This allows a system administrator to quickly provide resource access to various users based on the intended function.

The next field is the comment field. This field normally contains the user's full name, but a system administrator can really store any type of information they desire within this comment field.

The location of the home directory is the next field contained in the colon-delimited line. Every user account has an assigned home directory, which is an area where the user can store all their files, sub-directories, and individual configuration files. Just like how each user on a Windows system has a folder located at `c:\Users\<username>`, a Linux system maintains a similar area for the user's files. By default, this is in `/home/<username>`, but some distributions place this in a different location, such as `/Users/<username>`. A system administrator can set the location of a user's home directory to any location on the system, as well.

The final field is the location of the default shell assigned to the user account. Normally, this shell will be set to the bash shell located at `/bin/bash`. The user can configure the default shell for the account to whichever shell they prefer, including the Bourne Again Shell (bash), Korn shell (ksh), the TENEX C Shell (tcsh), and the Z shell (zsh).

Account Security

To secure a user's account, we should assign a long, strong password should. In earlier version of Linux, the password information was actually within the `/etc/passwd` file, but this practice was quickly modified since the `/etc/passwd` file is readable by all users. Therefore, storing a user's password in plain text in the `/etc/passwd` file would be extremely risky and a poor security practice.

Today, passwords are instead stored in the `/etc/shadow` file. This file cannot be read by ordinary users and is only accessible to the root user or by the system itself. Every user password is stored

as a salted hash digest. A salted hash uses a one-way mathematical algorithm with additional randomized input (known as the salt) to produce a secure method to store the password. When a user attempts to login into the system, the password they enter is salted and hashed, and then this result is compared against what is stored within the `/etc/shadow` file. If the input and the stored value match, the user can access the system.

Like the `/etc/passwd` file, the `/etc/shadow` file contains records for each user on the system, with each record containing one line of data with colon-delimited fields.

The first field is the username. Unlike the `/etc/passwd` file, the `/etc/shadow` file does not rely on UID to identify users. Instead, the username is identified using a unique alphanumeric field, such as `jason` or `diontraining`.

The second field contains the password, stored as a salted hash. Because it is a salted hash, the password doesn't bear any resemblance to the real password. If an asterisk (*) or exclamation mark (!) is in the password, this indicates that the account is locked. Most often, a locked account is a system account and is not directly logged into by an end user.

After the password field, the next field is the "Last Password Change". This field contains the date that the password was last changed, and other operating systems often refer to it as the password. The date is stored as a whole number, which is based on the number of days since January 1, 1970, not as a Julian calendar date.

The fourth field contains the "Days Until a Change Is Allowed". This value is stored as a whole number, and security professionals often use it to prevent a user from changing the password multiple times on the same day. Some system administrators will set this value to a non-zero value to prevent a user from changing their password and then immediately changing it back to their old password.

The "Days Before a Change Is Required" is the next field. This field dictates when the password must be changed again using a whole number that represents the number of days. For example, if the value is set to 7, then the password must be reset in one week (7 days).

The next field is the "Days of Warning Before Password Expiration". If this field is set, then the system is configured to

provide a warning to the end user that their password will need to be changed within the number of days set in the “Days Before a Change Is Required”.

In a Linux system, the system administrator can also configure the system to a specific period of time between an account’s expiration and actual deactivation. This field is called the “Days Between Expiration and Deactivation”, and it is the next field in the `/etc/shadow` file. When an account expires, it requires the user to either immediately change their password upon logging in or to contact a system administrator to unlock their account. If an account is deactivated, though, the user’s password is erased from the `/etc/shadow` file, and this requires a system administrator to manually reactivate the account.

To determine when an account will expire, see the “Expiration Date” field. This field shows the date on which the account will expire, and it is expressed as a whole number that’s the number of days since January 1, 1970. It is a security best practice to require every account to be used at least once every 60 days. If an account isn’t used within 60 days, it should be set as expired and then deactivated 30 days after that expiration.

The final field is the “Special Flag” field. This field is reserved for future use within Linux and is normally not used on modern Linux systems. If it is used, it generally contains a meaningless value that is unimportant to the user or system administrator.

Because of the password and security information contained within the `/etc/shadow` file, this file’s permissions are set to be highly restrictive and the file ownership is set to the root user. This keeps non-root users from accessing, reading, or modifying the file and prevents them from obtaining the passwords stored in their salted and hashed format. By contrast, the `/etc/passwd` file must be readable by ordinary users and therefore has less-restrictive permissions.

It is important to understand that a user account is not a singular thing like a file, though. An account is made up of lots of different pieces of information and configurations that are scattered across several configuration files, such as `/etc/passwd`, `/etc/shadow`, `/etc/group`, and other files that refer to each user account.

Understanding Groups

A group is a collection of user accounts that are defined within

the `/etc/group` file. Just like the `/etc/passwd` and `/etc/shadow` files, the `/etc/group` file contains all the records which define the groups on a system. Each record contains the information for one group listed as a colon-delimited line of four fields within the `/etc/group` file.

The first field, known as Group Name, contains the name of the group displayed in its alphanumeric form. Most commands use this field when identifying, accessing, or manipulating data about a group.

The second field is known as the password field and contains the password for the group if one exists. If a value of `x` is displayed, this indicates that there is a password, but it is stored elsewhere for increased security. If this field is empty, this indicates no group password.

This third field is the group identification (GID) number. Linux uses this identification to internally reference and identify each group. Just like with user identification numbers, group identification numbers are translated back to group names for the benefit of users and system administrators who prefer a human readable name, as opposed to a number.

The fourth and final field is the User List. This field specifies which users belong to the group. This field can contain one or more users. If multiple users are listed, each username should be separated by a comma within this field.

In order to identify a user as part of a group, the system can either identify the GID within the user's individual entry in the `/etc/passwd` file, or the system can specify the usernames in the User List field of a group's record in the `/etc/group` file.

Unfortunately, there is a limitation when identifying a group within a user's individual record in the `/etc/passwd` file. This method only allows us to assign a user to a single group within the GID field. If a group is associated with the user using this method, that group is the user's primary group.

Since a user can be a member of multiple groups, it is better to instead list the username within the `/etc/group` file under each group's record that they will be a part of. Each group that the user is added to using this method is a secondary group for the user.

Groups are an important part of assigning permissions to different files. Each time a new file is created, the file will be automatically associated with the user's current group. When a user

logs into a system, the user's current group is always set to their primary group by default. For a user to change their current group to one of their secondary groups, the user issues the `newgrp` command. To use this command, simply enter "`newgrp <group>`", where `<group>` is the name of the secondary group the user wants to use as their current group. After entering this command, any files created by the user will be associated with their new current group.

Account Management

There are numerous commands and utilities for manipulating user accounts and groups on a Linux system. For example, if a user needs to determine which user they are currently logged in as, the command `whoami` can be utilized. By typing `whoami` at the prompt, the system will display the current username. This is helpful when a user maintains multiple accounts on a system, such as a regular user and an administrator account, and needs to recall which account they are currently using.

If the user needs additional details or information, they should use the `id` command. By typing `id` at the prompt, the system will display the current UID, username, GID, and the user's group memberships to the terminal. The `id` command is extremely useful since it presents both the numeric UID and GID values, as well as their associated alphanumeric display names.

Because Linux is a multiuser operating system, not only does it support multiple users accounts, but multi users can also simultaneously be logged on to the computer. For example, my web server, diontraining.com, currently has thousands of active users who log in to our learning management system to take their online video courses at any given time. In our case, this is done via a web-enabled front end, but most Linux users and administrators instead log into a remote server using a remote access tool like Secure Shell (SSH).

Since multiple users cannot access the Linux system directly from the same keyboard, mouse, and monitor, Linux supports a feature known as a virtual terminal (VT). As a system administrator, it is important to know if there are other users who are currently logged on when you are performing system maintenance to prevent critical data from being lost. For example, if a system administrator installs a new software update or security patch, they may need to reboot the system. Prior to reboot the system, the administrator

should use the `who` command to determine who is currently online and logged in.

When `who` command displays a lot of valuable information to the terminal, including the username, terminal identifier, login data and time, and the remote host address.

The username is the first column and it shows the name of all of the users who are logged into the system.

The terminal identifier shows an alphanumeric code to indicate which terminal the remote user is on. If the terminal identifier is indicated as zero (0), then the user is logged in locally via the GUI. If the terminal indicator displays `pts/#` or `tty#`, this indicates a text session. Text session logins can be performed by launching a terminal within the GUI, through a text-mode console login, or remotely via SSH or another remote access protocol.

The Login Date and Time column indicates when the login was initiated. This can be helpful when determining if a connection is still active or if someone logged in many hours ago and forgot to log out at the end of their workday.

The Remote Host column indicates the source of the login. If the login was conducted locally via the GUI or the console, then this column will be blank. If the user opened a terminal from within the GUI, it is indicated as either `#` or `##`, with `#` being the number of the login source. If the login was made remotely, then a hostname or IP address will be listed under this column to indicate the source of the connection.

The `who` command is not the only method for identifying information about the logged in users, however. An alternative to `who` is `w`. The `w` command is similar to `who`, but it produces a more verbose output including the terminal identifier and login time in a different format. In addition to this, `w` also displays the session's idle time, the JCPU, the PCPU, and what is being run.

The session's idle time indicates how long it has been since the user has last interacted with the current session. This is extremely helpful when trying to determine if a user is actively using the system or simply forgot to log off at the end of the workday.

The JCPU column identifies the total amount of CPU time associated with the user's session. This information is used in some pay per use services, which are quite typical in cloud-based environments.

The PCPU column identifies the total amount of CPU time

associated with the current processes running in the session. This can be useful when troubleshooting a Linux system that is malfunctioning or acting slowly. By identifying which process and which user is overusing the CPU, the system administrator can eliminate this bottleneck and increase the overall system performance for all users.

The WHAT column indicates which program the session is running. Again, this is helpful in troubleshooting or identifying what each logged in user is doing on the given system.



To create a new user account from the shell, open a terminal and type "sudo adduser username". Replace username with the intended username. After hitting enter, the system will for a password for the username. Select a strong password. It will need to be retyped for the system to accept it. You may then be asked for the new account's user information such as the full name, room number, work phone, etc. Notice that it will also create a new group and add this user to that group, and it will also create the

user's home directory.

To verify that the user is now listed as one of the users on the system, type `cat /etc/passwd`. This will list all the users in that system. Since this user was just added, it should show at the bottom of the list. Even if this is the `passwd` file, it will not show the actual password, which is stored in a different file. Information shown from the `cat` command will show the user's UserID and Group number, information entered while creating the user account, home directory, and default shell. But on a system that may have thousands of users, there is a better way than manually searching. Typing `grep '^username' /etc/passwd`, replacing `username` for the user being searched, will generate the lines in the `etc/passwd` file that contain the username. To find out which user number the user is, type `grep '^username' -nu /etc/passwd`.

More information about the `adduser` command can be found in its man pages.

Another utility for creating user accounts is `useradd`. This is, however, more tedious than `adduser` and requires a longer string of commands. To add a user using `useradd`, type `sudo useradd -s default-shell -d home-directory -m -G group-name username`. You will be replacing `'default-shell'` with whichever default shell you want this user to use (usually, `/bin/bash`). Also, replace `'home-directory'` with the default home directory for the user, `'group-name'` for the group the user will be added to, and `'username'` with the actual username for the user account. One difference with `useradd` and `adduser` is that `useradd` creates the account but does not automatically ask for password assignment. To delegate a password to a user account created through `useradd`, type `sudo passwd username`. This command is also for changing the password of other user accounts.

Modifying and Deleting Accounts

Scan QR code to watch a video for this topic



Other than creating user accounts, system administrators will also need to manage these accounts, and a couple of the things that they need to do are modify and delete accounts. When modifying accounts, a common task is resetting passwords. To do that, type "sudo passwd username", replacing username with the user account whose password needs to be reset. Notice that there is no need to know the old password, as some of today's system require. This is because the root user has complete power to modify a Linux system.

The passwd command also has a lot of other features aside from setting a user's password. To get information about a user account in terms of password security, type "sudo passwd -S username", replacing username with the account's username. A line will show starting with the username and either a P, NP, or L. P after the username means that a password is good for this user; NP stands for no password; and L means that the user has been locked out of his account because he may have forgotten it and entered an incorrect password too many times. The next information on the line is the

date of the last password change. The next number indicates the minimum password expiry age and is usually set to 0, indicating that the user change it as many times as they want and as frequently as they want. The next set of numbers is the maximum age of the password in days, usually 99999. Then, the password expiry warning in days, usually 7. This means that 7 days before expiry, the user will be notified to change passwords. The last number is for the inactivity period of the password.

A better version of the "passwd -S" command and option is the chage command. Type "sudo chage -l username" to list information in a more readable format compared to the line listed by the passwd command above. Refer to the man page of chage for more information about the other options that allow changing password attributes such as the expiry age.

Aside from modifying passwords, the username may need to be modified either due to a misspelling or a user change altogether. Properly renaming a user requires a couple of commands: id and usermod/groupmod. To identify a user, type "id username". This will show the user id, group id, and the groups the user is a member of. The id command can verify the correct username and group assignment. To change the username, type "sudo usermod -l newusername oldusername", replacing newusername with the new username, while the old username replaces oldusername. After successfully changing the username, use the id command to check if the new username exists and that the previous group assignments remain. Take note, however, that usermod only changes the username. It does not change the group name nor the home directory. To change the group name, type "sudo groupmod -n newgroupname oldgroupname", replacing newgroupname with the new name of the group and oldgroupname with the group's old name. There are more options for modifying user and group attributes, which can be found in the man pages of usermod and groupmod.

There will also be certain instances we have to delete a user from the system, such as when an employee is no longer connected with a company. To delete a user, simply type "sudo deluser --remove-home username", replacing username with the user to be deleted. The '--remove-home' option removes the user's home directory, thereby deleting the files in it. Take note that, since this is done in superuser mode, changes may be irreversible, so be

cautious when deleting users and deleting user accounts and their files. Another way to accomplish user deletion is by typing "sudo userdel -r username". Once again refer to the man pages of deluser and userdel for more information about these commands.

The nice thing about Linux systems is that they keep a log of users created and deleted. The log could vary but for Ubuntu, it is located in /var/log/auth.log. This is the authentication log. The cat and grep commands can look for information in the auth.log file. For a recent event, such as a recently deleted user or group, type "tail -15 /var/log/auth.log" to see the last 15 lines of the log file, as that should be where the latest events are logged. For user deletion events, type "grep -E 'userdel' /var/log/auth.log" to check for the instances of the userdel command.



Previously, we talked about creating a user in the shell. Now it's time to discuss managing groups. Managing groups includes creating a group, adding people to groups, and renaming a group.

By default, in Ubuntu, whenever we create a user without

indicating a group, it automatically creates a matching group. So, typing "sudo useradd tim" creates the user tim and the group tim, of which the user, tim, is a member.

To create a new group, use the groupadd command. Type "sudo groupadd students" to create a group called students. To add the user tim which was created previously to the student group, type "sudo usermod -a -G students tim". The -a option means add this feature, while -G option means add this group.

To know which user is in which group, there is a file inside the etc folder called group that we can use with the grep command. Typing "grep students /etc/group" would show lines in the /etc/group file that contain the word "students". There will also be instances that you, as the system administrator, need to see the entire list of groups in the system. Typing "cat /etc/group |more" will show all the groups in the system, one page at a time. Hitting space on the keyboard brings the view to the next page.

To rename a group, type "sudo groupmod -n oldstudents students". The -n option indicates that the word after the option becomes the new name of the group, while the next word is the name of the target group to be changed.

Refer to the man pages of groupadd and groupmod for more information on these commands.

Working with the Root User

On every Linux computer, there is one user that has the extraordinary power needed to manage the whole system. This user is known as the root user, also known as the super user or administrator.

When most people use computers to do ordinary day-to-day computer tasks (known simply as user tasks), they don't require any special privileges. For example, if a user wants to create a new text document within their home directory, this is a user task and their user account has all the permission necessary to create the file.

The root user account, on the other hand, exists to perform administrative tasks such as installing new software, preparing a new disk, and managing all the other ordinary user accounts on the system. These administrative tasks require access to many different system and configuration files that ordinary users simply cannot access or modify.

There are three different ways to access the powers of the root

user. The first method is to simply login directly as the root user from the shell using the text-mode or by using a remote login tool such as SSH. Due to security restrictions within some distributions, though, this method may not be enabled.

The second method is to use the su command. The su (switch user or substitute user) command enables a user to change their user identity within the current shell. By typing su at the terminal, the user effectively becomes the root user and gains all their permissions. For this to work, the user must then enter the password for the root user, thereby proving that they have permission to use that account. Once the root privileges are acquired in this way, any subsequent commands typed into the shell will run as with the elevated privileges of the root user. To return to the status of a normal user, simply type exit at the terminal to relinquish superuser status.

The third method is to use the sudo command. The sudo (super user do) command is similar to the su command, but it only provides root privileges for a single command. For example, to edit the /etc/shadow file, a user may type “sudo nano /etc/shadow” to run the nano /etc/shadow command with root privileges. Using the sudo command is considered the best practice when operating using the root user’s permissions, since it only applies to a single command at a time.

When operating as the root user, we should take additional precautions. As Uncle Ben from Spiderman once said, “With great power comes great responsibility.” Nowhere is this truer than when operating as the root user.

Since a root user has permission to access, modify, and delete every file on the hard drive, one mistyped command as root could accidentally wipe out critical application files and cause hours of downtime on a Linux server. Conversely, if an intruder gains root access to a Linux system, that intruder can now make unintended changes or cause damage to the computer’s system files, change ownership or permissions on ordinary user files, or install backdoors and rootkits that allow them to continually maintain remote access to everything on that system. Therefore, it is imperative that users always take the following seven precautions whenever they consider using the root user’s permissions and access.

First, the user should ask they really need root access. Many

times, there is a better or more secure method of accomplishing the same goal without having to log in as the superuser.

Second, if a user is operating as the root user, it is imperative that they always double check the command before pressing the enter key. Whenever I am operating as root, I type the command, remove my hands from the keyboard, review the command, and verify it is completely accurate before pressing Enter. When operating as root, a simple typo or error can be completely catastrophic.

For example, let's assume that the user is in the `/home/jason/` documents directory and wants to delete all the files within the directory. The user wanted to enter the command `"rm -rf *.*"`, which would remove every file within the directory. But, the user mistyped the command as `"rm -rf /*.*"`, which would remove every file within the root of the file system instead. The simple addition of the `/` drastically changes the meaning of this command, and it could render the operating system completely unusable and unable to boot up.

Third, when operating as root, never run a suspicious program. Any program downloaded from the internet could be designed to compromise the security of your system. If you run the program as root, the program could then have full access to the entire system and not just the user's own account.

Fourth, whenever operating using root privileges, use those permissions for as short of a time as possible. Again, it is recommended to utilize the `sudo` command instead of using `su` or logging in directly as the root user. This will limit the length of time that the user is operating as root to a single command only.

Fifth, never leave a shell that is operating as the root user accessible to others. When performing maintenance tasks that require the root user's permissions, always enter `"exit"` in the root shell prior to walking away from the system.

Sixth, always use a long, strong password for the root account. The root account's password must be well-protected, and the password should never be reused on any other user account.

The seventh and final precaution is to never share the root password with other users. If working in a public area, be cautious and ensure that no one is looking over their shoulder when entering the root password.

The root user and its permissions provide the ultimate power

for a user or system administrator on any Linux system. For this reason, it is important to always ensure the account remains protected and its utilization carefully monitored.

CHAPTER TWELVE

Ownership and Permissions

```
# ls -l file
-rw-r--r-- 1 root root 0 Nov 19 23:49 file
```

Other (r--)	r = Readable
Group (r--)	w = Writeable
Owner (rw-)	x = Executable
	- = Denied

File type

OBJECTIVES

- Be able to set the ownership of a files and directories
- Understand and calculate permissions for files and directories
- Explain sticky bits and special execute permissions
- Be able to configure permissions for files and directories

Due to its status as a multiuser operating system, Linux provides tools to enable users and system administrators to secure files and directories against unwanted access. To secure files and directories on a system, Linux assigns ownership and permissions to each and every one.

Ownership and permission are intertwined, since each file is associated with the owner of the file and the group of which the owner is a member. Permissions are then set based on what the owner of the file can do to the file, what other members of the owner's group can do to the file, and what other users on the system can do to the file. Linux uses different tools for manipulating ownership and manipulating permissions of a file or directory. Let's consider the concept of ownership in more detail and the commands used to manipulate that ownership.

Ownership

Linux's security model assumes the presence of multiple users on the computer and provides the means to associate individual files with the users who create them. Each file or directory within a Linux system is assigned a user account and group as an owner. By default, when we create a file, it uses the currently logged in user and their current group.

Each program or process in a system also has ownership assigned to it. When a program executes and a process launches, the user who performed that action is assigned the owner by default. This ownership identity is considered, along with the file's ownership and permissions, to determine whether a program may modify a given file.

For example, if the user `diontraining` attempts to execute the command `"nano /etc/shadow"`, it will generate an error since `diontraining` doesn't have sufficient permissions to access or modify the `/etc/shadow` file. Since the `nano` command was run by `diontraining`, the command was assigned ownership to `diontraining` and inherited its permissions. To overcome this ownership limitation, `diontraining` could instead execute `"sudo nano /etc/shadow"`, which executes the `nano` program using the root permissions. In this example, the `nano` program is assigned ownership to the root user, and inherits its permissions, which includes the ability to access and modify the `/etc/shadow` file.

Since every file has an owner, each file becomes associated with a user account's UID number. Additionally, every file is also associated with a group through its GID number. This allows the system to control access to each file using permissions that can be set independently for the file's owner, the file's group, and all other users of the computer.

To fully control the ownership of a file or directory, utilize the root user. The root user can change the ownership of any file or directory by using the command `chown` within a terminal. Execute the `chown` (change owner) by typing `"chown <username> <file or directory>"`. To learn more about the `chown` command, please type `man chown` within the terminal.

The owner can change the group associated with any of the files or directories that they own, but they can only change it to other groups to which the user is a member. If the file or directory needs

to be changed to a group that the owner is not a member of, then the root user account must make this change instead. To change the group associated with the ownership of a file or directory, use the `chgrp` command by typing “`chgrp <username> <file or directory>`”. To learn more about `chgrp` command, please type `man chgrp` within the terminal.

Permissions

File ownership is meaningless without providing a way to control what can actually be done with each file or directory. Permissions are set in three groups known as owner permissions, group permissions, and world or other permissions. Owner permissions dictate what the owner of a file or directory may do with that file. Group permissions dictate the actions that members of the owner’s group can perform. The final category of permission, known as world or other, indicate what actions can be performed by any other user on the system who is not the owner or a member of the owner’s group.

To determine a file’s permissions, a user can create a long directory listing that will display the permissions for the file using the command `ls -l <file>`. Consider the following example which lists the permission for the file known as `test.sh`:

```
$ ls -l test.sh
-rwxr--r-- 1 jason instructors 45 Nov 19 23:49 test.sh
```

The output of the long directory listing includes the permission, the number of links, the username, the group name, the file size, the time stamp, and the filename.

The first column displays the permissions for the file, in this case

`-rwxr--r--`. The first `-` indicates the file type code. This single character code of `-` indicates this is a normal data file that may be text, an executable program, graphics, compressed data, or any other type of data. If this first character is a `d`, this indicates the file is a directory. If it is a `|`, it indicates it is a symbolic link to a file or directory. If it is a `p`, it indicates a named pipe that enables two running Linux programs to communicate with each other in one-way manner. If it is a `s`, it is a named socket that permits network and bidirectional communication. If it is a `b`, it indicates the file corresponds to a hardware device which uses blocks to transfer data, such as a hard disk drive. If it is a `c`, it indicates a character

device, which is a file that allows a single byte of data to be transferred at a time (for example, a parallel or serial port). In general, most files will have either a `-` or a `d` as their file type code.

The next 9 characters of the permission string is used to represent whether this file can be read (`r`), written to (`w`), or executed (`x`), and by whom. The first three of these characters define the permission for the user, the next three for the group members, and the final three are for the world or other permissions.

After the permission string, the next column represents the number of hardlinks on the system for this file. In that case of the `test.sh` file, only a single hardlink exists on the system. Remember, a hardlink is a unique filename that may be used to access this file, therefore in this case only `test.sh` is used to reference this file.

The next column contains the file owner's username. In this example, the file `test.sh` is owned by `jason`. This file is also associated with the group name of `instructor`, which is the next column displayed in the output.

After the username and group name, the size of the file in bytes is displayed. Since this is a short and simple script, the file size is only 45 bytes.

The next column contains a time stamp. This identifies the time that the file was last modified. In this case, the file was modified last on Nov 19th at 11:49 pm (23:49).

The final column contains the filename. While this isn't as useful in this example since we entered the filename when we initially executed the command, it is useful if the contents of a directory are being displayed instead. For example, if the user simply typed `"ls -l"`, then every file and sub-directory within the current directory would be listed and the filename field identifies each one.

To change the permissions on a file, use the `chmod` command `chmod`. For example, to set the read, write, and execute permissions for users, groups, and world/other for the `test.sh` file, the user will enter `"chmod ugo=rwx test.sh"`. The `u` (user), `g` (group), and `o` (world/other) identify which permissions are used for which.

Octal Permissions

In the previous section, we described the four parts of the permission string: file type code, user permissions, group permissions, and world or other permissions. Each of these three

permission groups were represented with the letters r (read), w (write), or x (execute).

Read permissions simply indicate whether one can actually open up the file to see the contents. Write permissions indicate that the user can modify the file or create new files within the directory. The execute permission, though, indicates if a file may be run as a program or script. If the user, group, or world/other does not have permission to take the read/write/execute action, its corresponding location in the permission string is replaced with a - instead of its symbolic letter.

For the LPI Linux Essentials exam, it is important that you can read and understand what a permissions string indicates. For example, if the permission string of -rwxr-xr-x was displayed, what does it indicate to you about the file's permissions?

First, the file type code is a -, which indicates it is a normal data file.

Second, the first three characters of rwx indicates that the user can read, write, and execute this file.

Third, the second three characters of r-x indicates that other members of the user's group can read and execute the file.

Finally, the third three characters of r-x indicates that the world or other users on the system can read and execute the file.

In summary, this file is a normal data file that the user who owns it can read, write, and execute, and every user can read and execute it. Being able to understand and simply explain the permission strings is crucial to your success on the LPI Linux Essentials exam.

Now, there is a second way, known as octal, to indicate permissions within Linux. An octal is a shorter way of writing the permission strings by using three numbers from 0 to 7 to indicate the permission for the user, group, and world/other permissions.

Each number in the octal is created by assigning a value to the read, write, and execute permissions. If read permissions are set, then a 4 is given. If write permissions are set, then a 2 is given. If an execute permissions are set, then a 1 is given. If multiple permissions are given, then the values are added together.

For example, if the file has read, write, and execute permissions, this is set to 7, because read (4) plus write (2) plus execute (1) equals 7.

If a file has read and write permissions but no execute

permissions, then the value is 6. If a file has read and execute permission, but no write permissions, then the value is 5. If no permissions are given, then the value is 0. If it has write and execute permission, then the value is 3.

Each number represents a single three-character permission set. To assign permissions for the user, group, and world/other, use three digits. Returning to the earlier example of `-rwxr-xr-x`, what would the octal permission set become?

The answer is 755. This is because the user can read (4), write (2), and execute (1); therefore, its value is 7. Then, the group can read (4) and execute (1); therefore, its value is 5. Like the group, the world/other can also read (4) and execute (1); therefore, its values is 5.

To change the permissions on a file, use the `chmod` command. For example, to set the read, write, and execute permissions for users, groups, and world/other for the `test.sh` file, the user will enter “`chmod 777 test.sh`”.

Special Cases with Permissions

There are a few special cases to permissions of which a user should be aware. The most important is the fact that most permission rules simply don't apply to the root user. Remember, a superuser can read or write any file on the system, including ones that have their permission set to 000.

While the Linux filesystem treats files and directories fairly equivalently in setting permissions, it is important to note that those permissions mean slightly differing things when used on a file and a directory.

If a file is granted execute permission, it indicates that the file can be run as a program or script. If a directory is granted execute permission, though, it indicates that the directory may be searched using various commands.

If a file is granted write permission, this indicates that the file may be modified or overwritten. Since directories are simply files that are interpreted in a special way, though, a directory with the write permission set allows users to create, delete, or rename files in the directory. It is important to note that, if a user has write access to a given directory and even if the user isn't the owner of the files within the directory and does not have permission to write to the those file, they can still delete or rename the files.

This may seem like a bug — after all, if a user can't write to a file, then it would seem logical to think that they shouldn't be able to delete that file. However, it is important to remember that directories are just a special type of file, a file that holds other files' names and pointers to their lower-level data structures. So, while modifying a file requires write access to the file, creating or deleting a file only requires write access to the directory in which it resides. Therefore, this is not a bug in Linux; it is just a counterintuitive feature of the Linux filesystem and its permissions.

Finally, symbolic links are a special case in terms of permissions. This is because symbolic links always have their permission set to 777 and have read, write, and execute permissions enabled for all users on the system. The 777 permission only applies to the symbolic link itself and not to the file that is being linked to. That linked file retains whatever permissions have already been set for it.

Setting the User Mask

The user mask is used to determine the default permissions that will be assigned to all newly created files. The `umask` command adjust the user mask for the system. By default, new files have the user mask set to 666 (-rw-rw-rw-) and new directories have the user mask set to 777 (-rwxrwxrwx).

The `umask` command is usually used within a system script or configuration file. To learn more about `umask`, please enter `man umask` within the terminal.

Sticky Bits

Although Linux filesystems were designed to work as described above, this behavior is not always desirable. In order to create a more intuitive result, we use sticky bits. A sticky bit is a special filesystem flag that alters a behavior. When we set the sticky bit on a directory, Linux permits a user to delete a file only if the user owns either the file or the containing directory. Simply put, when using a sticky bit, the user cannot delete a file simply by having write permissions to the containing directory.

To set the sticky bit, use the command `chmod` with a special octal code or a symbolic character code. To use an octal code with `chmod` to set the sticky bit, the three-digit octal code must be prefixed with a 1 (to enable the sticky bit) or a 0 (to remove the

sticky bit). For example, if we enter the command “`chmod 1777 test.sh`”, this will set the file `test.sh` to read, write, and execute for every user on the system and enable the sticky bit.

The second method is to set the sticky bit using a symbolic character code. This adds the symbolic code of `t` to the permission string using `chmod`. For example, if we enter the command “`chmod ugo=rwx+t test.sh`”, this will set the file `test.sh` to read, write, and execute for every user on the system and enable the sticky bit. To remove the stick bit, simply use `-t` within the command instead of `+t`.

Special Execute Permissions

As previously discussed, the executive permission bit enables a user to identify which files are considered programs and scripts. When this execute bit is included in the permissions for a file, it indicates to the Linux system that the file is an executable file and should be considered a program file (providing it the same capabilities as a Windows’ file with the extension of `.exe`).

By default, files execute using the credentials of its file owner. This allows Linux to associate specific users with different running processes and is a key security feature of the Linux operating system. Occasionally, though, a program will need to run with elevated permissions, such as the superuser or root.

For example, the `passwd` program sets user passwords on the system. This program must be run as the root user to write to the configuration files that store the passwords, such as `/etc/shadow`. So, if a user needs to change their own password using the `passwd` program, then `passwd` must have root privileges, even when an ordinary user is the one who executes the program.

But, as pointed out earlier, the root password should not be given to other users on the system due to security concerns. Luckily for us, Linux provides us with two special permission bits that can help us solve this challenge. These special permission bits operate much like the sticky bits discussed earlier in this chapter.

The first special execute permission bit is known as SUID (Set User ID). The SUID option indicates to the Linux system that the file is to run with the permissions of whoever owns the file, rather than with the permissions of the user who actually began running the program.

For instance, if the root user owns the `passwd` program file and

has set a SUID bit, the program can be executed by any normal user and automatically runs with root privileges. This allows the program to read any file on the computer because the program is operating as the superuser. Some server programs and daemons operate in this way. When a program operates under this model, it is referred to as being SUID root. To identify a SUID program on your system, look for the `s` in the owner's execute bit position within the permission string. For example, the following permission string indicates SUID root for this file: `-rwsr-xr-x`.

The second special execute permission bit is as SGID (Set Group ID). The SGID is similar to the SUID option, but it operates by setting the running program's group to the file's group. The SGID is indicated by a `s` in the group execute bit position of the permission string, such as in `-rwxr-sr-x`. Notice the key distinction here; the SUID operates as the root user, but the SGID operates as the group's user instead.

To set the SUID or SGID, use the `chmod` command with either an octal code or a symbolic code. To set the SUID using an octal code, set the first digit to a 4. To set the SGID using an octal code, set the first digit to a 2. To set both the SUID and SGID, set the first digit to a 6. For example, if we issue the command "`chmod 4755 test.sh`", then the file `test.sh` will have the SUID bit set and permissions of `-rwsr-xr-x`.

To set the SUID or SGID bit using a symbolic code, use the letter `s`. For example, if we enter the command "`chmod u+s test.sh`", this will enable the SUID bit for the file `test.sh`. For example, if we enter the command "`chmod g+s test.sh`", this will enable the SGID bit for the file `test.sh`. To set both the SUID and SGID, use "`chmod ug+s test.sh`". To remove the SUID or SGID bits, use `-s` within the command instead of `+s`.

In general, most users will never have to set or remove the SUID or SGID bits themselves, since most package management programs will set or remove these bits for the user during the installation, upgrading, or removal of a program.

Hiding Files

Windows users may be familiar with the concept of a hidden bit, which hides files from view in the graphical user interface file managers and from the `dir` command. While Linux doesn't have this as a dedicated filesystem feature, there is a special file-naming

convention that is hide files from view: a single dot (.) prefix. By adding this prefix, most tools will simply hide the files and directories from view anytime their name begin with a single dot. Be aware, though, that while renaming a file with a single dot prefix will hide it, this action will make the file inaccessible to any program that uses the original filename.

For example, if there are two files in the current directory called `myfile.txt` and `.myfile.txt`, and a user executes the `ls` command, then only the file `myfile.txt` will be displayed while the `.myfile.txt` remains hidden from view.

Most file managers and dialog boxes that deal with files also hide these so-called dot files, but this practice is not universal across every Linux program.

Due to this simple convention, many user programs will hide their configuration files from cluttering up the display by simply adding a dot to the front of the filename. Depending on the program in question, there are various ways to view these hidden files. In some GUI tools, there is a check box that can be set in their preferences or configuration options to force the program to display all files, including the hidden ones.

Within the command line, adding the `-a` option to the `ls` command will display the hidden files. For example, most system administrators commonly use `ls -la`, since it will show all the hidden files and directories, as well as their permissions.

When the command `ls -a` is executed, the user will notice two hidden directories contained within every directory on a Linux system. These are known as dot (.) and dot dot (..).

Dot (.) refers to the current directory. This is also known as the present working directory. Personally, I always refer to it as “here”. Therefore, if you want to create a relative path, you can use the single dot as your starting point within a command or file’s paths.

The dot dot (..) refers to the parent directory. This is the directory one level above the current directory. Again, this is used heavily when creating relative paths to different files and directories across the filesystem.

CHAPTER THIRTEEN

Conclusion



OBJECTIVES

- Take and pass the LPI Linux Essentials certification exam

Congratulations! You've made it to the end of this book, but there is still more to do. I know that we have covered a lot of material in this book, from getting started with Linux, to installing your own Linux distribution, to working in the command line, and even setting configuring permissions and ownership of files and directories.

I know that you are probably excited to move on and take the certification exam, but first we want to make sure that you are fully ready. I recommend that you take practice exams so that you are sure that you are good to go before the big exam. I have included one practice exam in this book, but feel free to take as many as you want—you can find others on the internet, throughout other texts, and other resources. If you have taken the time to register this book, you can find many additional practice exams included in our companion course.

If you take my practice exam and score at least 85%, you are probably ready to take (and pass) the LPI Linux Essentials certification exam. Nonetheless, take a couple of practice exams so

that you feel comfortable before test day, I promise it will help. Practicing will ease your nerves, give you the confidence needed to succeed, and make sure that everything is sticking properly in your head, so that you can pass the test the first time.

Your journey to becoming a certified Linux user and system administrator starts now. Good luck!

CHAPTER FOURTEEN

Practice Exam



GUIDELINES

- Practice exams should be taken without using any notes
- Complete the 40 questions in under 60 minutes
- Achieve a score of 85% or higher on this exam

1. Which of the following is an example of embedded Linux?
 - a) Android
 - b) Red Hat Linux
 - c) Fedora
 - d) Linux Mint
2. Which Linux distribution is used as a basis for the creation of Ubuntu Linux?
 - a) Arch
 - b) SUSE
 - c) Gentoo
 - d) Debian
3. Which of the following programs on a Linux system could you use as a replacement for Microsoft Word?
 - a) Pages
 - b) Pico
 - c) Writer
 - d) Nano
4. Which of the following programs on a Linux system could you use instead of Windows Media Player?
 - a) VLC
 - b) QuickTime
 - c) iTunes
 - d) Winamp

5. Which of the following is TRUE about open source software?
- a) Open source software is always free
 - b) Open source software is a prominent example of open collaboration
 - c) Open source software is more secure than closed source software
 - d) Open source software does not have any support
6. What does GUI stand for?
- a) General User Interface
 - b) Great User Image
 - c) Generic Unique Interlink
 - d) Graphical User Interface
7. Which of the following is NOT a Linux desktop environment?
- a) KDE
 - b) XFCE
 - c) Dwarf
 - d) Gnome
8. The output of the program `date` should be saved in the variable `datenow`. What is the correct statement to enter into the shell to set this variable?
- a) `set datenow = 'date'`
 - b) `date | datenow`
 - c) `datenow = 'date '`
 - d) `date > $datenow`

9. Which of the following is NOT a valid rule for naming a variable?
- a) Variable names must start with a letter
 - b) Variable names must start with numbers
 - c) Underscores are used instead of embedded spaces
 - d) Punctuation marks are not allowed
10. To see all the variables that are in the user's environment, use the _____ command.
- a) printer
 - b) ping
 - c) printenv
 - d) processvar
11. What could provide a set of electronic pages that explains a specific command available on the system?
- a) Man pages
 - b) Help pages
 - c) Master pages
 - d) Informational pages
12. Which of the following are not man page sections?
- a) Name
 - b) Mistakes
 - c) Configuration
 - d) Return Value

13. How can the current directory and its subdirectories be searched for the file named MyFile.xml?
- a) `find . -name MyFile.xml`
 - b) `grep MyFile.xml | find`
 - c) `grep -r MyFile.xml`
 - d) `find -P MyFile.xml | grep`
14. The "cd" command is short for _____.
- a) Check Directory
 - b) Change Directory
 - c) Change Disk
 - d) Check Disk
15. Which command is used to copy files from one directory to another?
- a) `mv`
 - b) `copy`
 - c) `cp`
 - d) `rm`
16. What does the "rm -rf" command do in Linux?
- a) Remove all the files in the directory and subdirectories
 - b) Remove all the files in the current directory
 - c) Rename all the files in the directory and subdirectories
 - d) Rename all the files in the current directory

17. Which of the following commands will create an archive file, named backup.tar, containing all the files from the directory /home?

- a) tar -cf backup.tar /home
- b) tar -cf /home backup.tar
- c) tar -xf /home backup.tar
- d) tar -xf backup.tar /home

18. Which of the following commands creates an archive file work.tar from the contents of the directory ./work/?

- a) tar --new work.tar ./work/
- b) tar --cf work.tar ./work/
- c) tar --create work.tgz --content ./work/
- d) tar work.tar < ./work/

19. How can the normal output of a command be written to a file while discarding the error output?

- a) command < output > /dev/null
- b) command > discard-error > file
- c) command > /dev/null 2>1 output
- d) command > file 2>/dev/null

20. Which command will display the last line of the file document.txt?

- a) head -n 1 document.txt
- b) tail -n 1 document.txt
- c) tail document.txt
- d) last -n 1 document.txt

21. Why is the file data.txt empty after executing `sort data.txt > data.txt`?

- a) Because sort cannot sort text files, only binary files
- b) Because sort detects that both files are the same
- c) Because the file gets truncated before sort is executed
- d) Because the file gets executed before sort is truncated

22. What is the output of the following command?

```
for token in a b c; do  
    echo -n ${token};  
done
```

- a) anbn cn
- b) abc
- c) \$token\$token\$token
- d) {a}{b}{c}

23. Which of the following statements may be used to access the second command line argument to a script?

- a) \$arg2
- b) \$2
- c) \$1
- d) \$var2

24. What keyword should be used to fill in the blank in the following segment of the given shell script?

```
for i in *; ____  
cat $i  
done
```

- a) do
- b) then
- c) when
- d) fi

25. How is it possible to determine if an executable file is a shell script which is read by Bash?

- a) The file must end with .sh
- b) The first line starts with #!/bin/bash
- c) /bin/bash has to be run in debug mode
- d) Only if you are logged in as root

26. Which of the following is NOT TRUE about the Linux Distribution Life cycle?

- a) Linux distros are always short term releases
- b) Most of the release schedules are publicly announced months or years in advance
- c) Some developers come up with catchy names for versions
- d) The final-release version is considered to be the stable version

27. What does HCL mean?

- a) Hindustan Computer Limited
- b) Hardware Compatibility List
- c) Hardware Computer License
- d) House Computer Linux

28. Which of the following properly identifies the third partition, on the second hard disk, on the first IDE controller on a PC system?

- a) /dev/hdb3
- b) /dev/hd1b3
- c) /dev/hdc1d2p3
- d) /dev/hdc1a3

29. After installing a new package, in which directory are you most likely find its configuration file?

- a) /lib
- b) /etc
- c) /conf
- d) /usr

30. Which of the following directories is often used to store log files?

- a) /var
- b) /temp
- c) /dev/hdc1d2p3
- d) /usr

31. What is the command that will show system boot time messages?

- a) echo
- b) lspci
- c) dmesg
- d) display system boot

32. Which network interface always exists in a Linux system?

- a) lo
- b) eth0
- c) wlan0
- d) vlan0

33. Which of the following displays network connections for Transmission Control Protocol, routing tables, and a number of network interface and network protocol statistics?

- a) net
- b) ping
- c) netstat
- d) traceroute

34. Which of the following commands can determine the time of the last login of a given user?

- a) showlog
- b) recent
- c) last
- d) history

35. Which command shows who is logged on and what they are doing in the system?

- a) who
- b) w
- c) whoami
- d) id

36. Which of the following files holds the definition of the local user accounts?

- a) c/users
- b) /etc/passwd
- c) c/id
- d) c/home

37. Which command adds the new user tux and creates the user's home directory with default configuration files?

- a) usercreate tux
- b) useradd -o default tux
- c) useradd -m tux
- d) passwd -a tux

38. Which of the following commands will display a list of all files in the current directory, including those that may be hidden?

- a) ls -a
- b) ls -h
- c) ls -all
- d) ls -hidden

39. What does the letter t at the end of drwxrwxrwt indicate within the following directory permissions? drwxrwxrwt 14 root root 36864 2012-03-02 11:17 /tmp

- a) The t is used to show that the directory is globally writable, but only the owner can delete their own files within the directory
- b) The directory is accessible by everyone
- c) The directory contains only temporary files that can be deleted by the system when more space is needed
- d) The files contained in this directory can only be access by the root user

40. What command would you use to create a symbolic link in Linux?

- a) ln -d
- b) ln -s
- c) link -d
- d) link -s

APPENDIX A

Answer Key to Practice Exam



OBJECTIVES

- Understand the correct answers to the Practice Exam

As you check the answers to your Practice Exams, it is important to understand why each answer is correct. As you go over your practice exam results, ensure that you pay close attention to the questions you missed and that you understand the explanation provided for their correct answers.

1. **A** Android, a Linux kernel-based operating system acquired and extended by Google, has become a highly competitive platform used in smartphones, tablets, and home appliances such as TVs.
2. **D** Ubuntu is a free and open source Linux distribution based on Debian.
3. **C** Writer is a well-known word processor that is part of the LibreOffice suite. It can be used for just about anything that Microsoft Word can do. While vi and nano are text editors, they are not full featured word processors like Microsoft Word. Pages is a word processor developed by Apple, and it only works on Mac OS X systems.
4. **A** VLC is a media player made by the VideoLAN project. It is cross-platform and works great on Linux systems for playing all types of video and music files. The other options provided only work on Mac OS X or Windows systems.
5. **B** Open source software is usually developed in a collaborative public manner and is a good example of open collaboration.
6. **D** The graphical user interface is a form of user interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, instead of text-based user interfaces, typed command labels, or text navigation.
7. **C** Linux has numerous desktop environments. Some of the more popular desktop environments are: Cinnamon, Gnome, KDE, XFCE, Unity, and Mate.
8. **C** To store the output of a command in a variable, you can use either of the following commands in the shell: (1) `variable_name=$(command)` or (2) `variable_name='command'`.
9. **B** The name of a variable should only contain letters (a to z or A to Z), numbers (0 to 9) or the underscore character (_). Variables should also always start with a letter.
10. **C** The `printenv` command displays the values of environment

variables.

11. **A** The man pages are a set of pages that explain every command available on the system. This includes information on what they do, the specifics of how to execute them, and what command line arguments or syntax they accept.
12. **B** There is no Mistakes section. It should be Errors.
13. **A** find is a command that searches one or more directory trees of a file system, locates files based on some user-specified criteria, and applies a user-specified action on each matched file.
14. **B** The cd command stands for Change Directory.
15. **C** The cp command stands for copy.
16. **A** The rm command is a UNIX and Linux command line utility for removing files or directories on a Linux system. When you combine the -r and -f flags, it means that recursively and forcibly remove a directory (and its contents) without prompting for confirmation. You should always keep in mind that “rm -rf” is one of the most dangerous commands that you can ever run on a Linux system, especially as root.
17. **A** The tar command allows you to quickly access a collection of files and place them into a highly compressed archive file. The -c option creates a new archive while -f uses the given filename as the archive's filename. Specifying the directory ensures the files to be placed in the archive file come from that folder.
18. **B** The tar command allows you to quickly access a collection of files and place them into a highly compressed archive file. The -c option creates a new archive while -f uses the given filename as the archive's filename. Specifying the directory ensures the files to be placed in the archive file come from that folder.
19. **D** Specifying 2> /dev/null will filter out the errors so that they will not be output to your console. In more detail: 2 represents the error descriptor, which is where errors are written to. By default, they are printed out on the console. /

dev/null is the standard Linux device where you send output that you want to ignore.

- 20. **B** The tail command with the -n 1 option will show the last line of a file.
- 21. **C** This is because the redirection is carried out first. This means that the > data.txt truncates the file so that sort finds nothing.
- 22. **B** A token is a sequence of characters that forms a word or punctuation sign. It is made up of ordinary characters or operator characters, but not both.
- 23. **B** Inside a script, the \$1 variable references the first argument from the command line, \$2 is the second argument, and so on. The variable \$0 references the current script.
- 24. **A** The for loop lets you iterate or 'do' over a series of 'words' within a string.
- 25. **B** The first line of the script should start with #!/bin/bash. Most shell scripts will end with a .sh by convention, but it is not required. Remember, in Linux, file extensions are only useful to the end user, but they are completely ignored by the operating system.
- 26. **A** There are distros that offer short-term support versions and long-term support versions or LTS.
- 27. **B** A hardware compatibility list or HCL lists tested, compatible, and sometimes incompatible hardware devices for a particular distro.
- 28. **A** Partitions on these disks can range from 1 (for the first partition) to 16 (for the sixteenth partition). If you have a few ide/pata devices installed in a system, they will be hda (the first device), hdb (the second device), hdc (the third device), and so on. Each partition on that device will append a number to it, so the third partition on the third device would be /dev/hdb3.
- 29. **B** /etc is a folder which contains all your system configuration files.

- 30. **A** /var is a standard subdirectory of the root directory in Linux that contains files to which the system writes data during its operation. /var/log contains log files.
- 31. **C** dmesg, short for display message or driver message, is a command that prints the message buffer of the kernel. The output of this command typically contains the messages produced by the device drivers that occur during the system startup and boot.
- 32. **A** lo is the loopback interface. This is a special network interface that the system uses to communicate with itself, and it exists even if there is no hardware network interface (like eth0 or wlan0) in the system.
- 33. **C** netstat is a command that displays network connections for Transmission Control Protocol, routing tables, and a number of network interface and network protocol statistics.
- 34. **C** The last command reads from a log file and prints the entries of successful login attempts made by the users.
- 35. **B** The w command displays a list of all logged in to the system and what they are doing. This command is similar to the who command, but it ends up displaying more information about logged in users.
- 36. **B** /etc/passwd contains the attributes (i.e., basic information about) of each user or account.
- 37. **C** The useradd command with the -m option creates the user's home directory with default configuration files.
- 38. **A** The ls command stands for list. The -a option lists all the files in the current directory including those that may be hidden.
- 39. **A** The t is a sticky bit, a specialized permission bit that is set on a directory that allows only the owner of the file within that directory, the owner of the directory, or the root user to delete or rename the file.
- 40. **B** To create a soft link from the shell, you should use "ln -s <original_file> <link_created>". The soft link is essentially

a pointer to the original file, and when the original file is deleted the soft link does not point to anything, and so “no such file or directory” is reported.

LPI® LINUX ESSENTIALS

Exam Vouchers



www.diontraining.com/vouchers

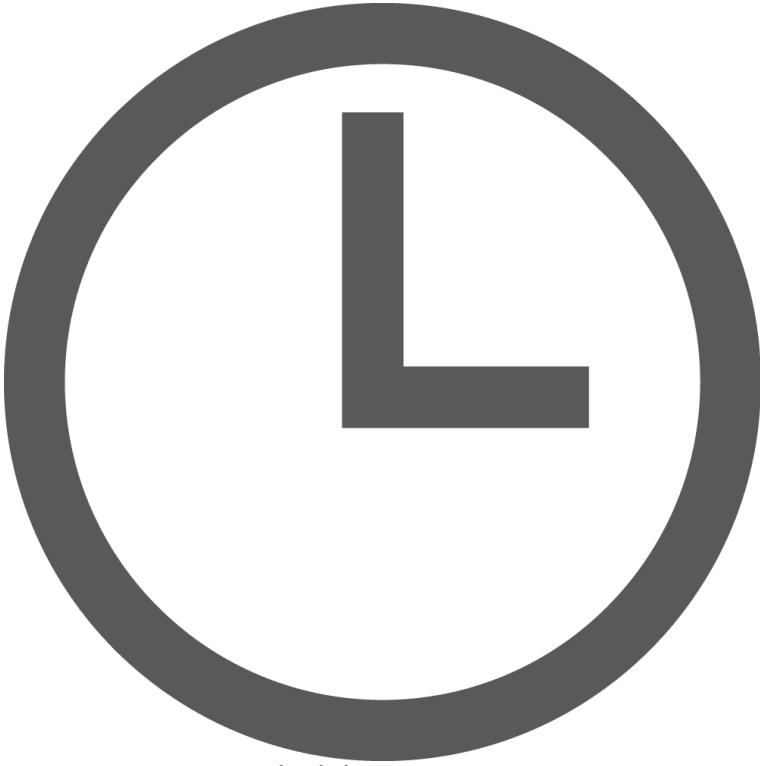
Save Money

The only difference between our exam vouchers and the one you buy at LPI.com or PearsonVue.com is the price. Since we buy thousands of vouchers per year, we receive a discounted price, and we pass the savings on to you.

SALE

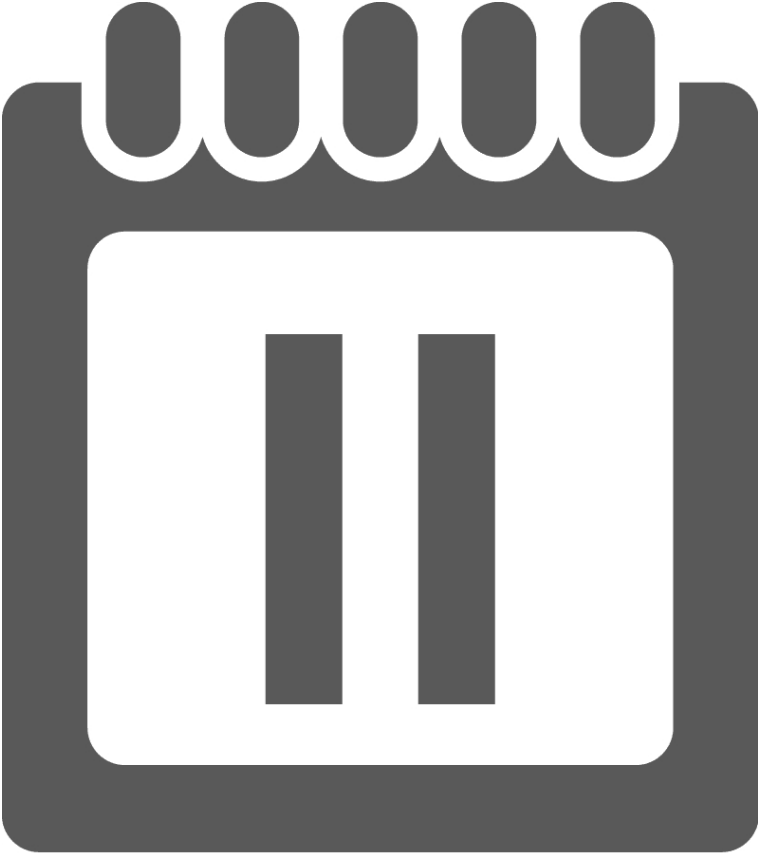
Fast Delivery

Our exam vouchers are delivered straight to your email within 15 minutes of your purchase, unlike other companies that take up to 24-48 hours to deliver your exam voucher.



Schedule Your Exam

Our exam vouchers are valid for 9 months from your date of purchase. They can be used to schedule your exam at your local PearsonVue test center or using their online web-proctored exam (OnVUE) any time of the day or night, 7 days a week, 24 hours a day, 365 days a year!



ABOUT THE AUTHOR



Jason Dion, is the lead instructor at Dion Training Solutions and a former college professor with University of Maryland University College, Liberty University, and Anne Arundel Community College. He holds numerous information technology professional certifications, including Certified Information Systems Security Professional (CISSP), CompTIA PenTest + , CompTIA Cybersecurity Analyst + (CySA +), CyberSec First Responder (CFR), Certified Ethical Hacker (CEH), Certified Network Defense Architect (CNDA), Digital Forensic Examiner (DFE), Digital Media Collector (DMC), CompTIA Security + , CompTIA Network + , CompTIA A + , ITIL® Managing Professional, PRINCE2® Practitioner, and PRINCE2® Agile Practitioner.

With information technology and networking experience dating back to 1992, Jason has held positions as an IT Director, Deputy Director of a Network Operations Center, Network Engineer, and numerous others. He holds a Master of Science degree in Information Technology with a specialization in Information, a Master of Arts and Religion in Pastoral Counseling, and a Bachelor of Science in Human Resources Management.